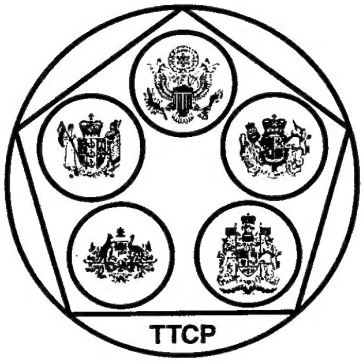

THE TECHNICAL COOPERATION PROGRAM

SUBCOMMITTEE ON NON-ATOMIC MILITARY RESEARCH AND DEVELOPMENT

Trends and Applications in Common Operating Environments and Distributed Computing Environments

2nd Edition with COE Security Appendix

Workshop Report from TTCP C3I TP-10

Published by

The Defence Evaluation and Research Agency
United Kingdom

April 2000

THE TECHNICAL COOPERATION PROGRAM
(TTCP)

Trends and Applications in Common Operating Environments and Distributed Computing Environments

2nd Edition with COE Security Appendix

The Report from
the 1996/1997 TTCP C3I TP-10 Study and Workshop

Published by

The Defence Evaluation and Research Agency
United Kingdom

April 2000

PREFACE

On 17 - 21 February 1997 the US hosted a TTCP C3I TP-10 Common Operating Environment Study Workshop at the US Navy Naval Command, Control and Ocean Surveillance Center (NCCOSC) Research, Development Test and Evaluation Division (NRAD) San Diego, California.

Following the Workshop and at the 2nd Plenary meeting of TP-10 held at CRC/DREO Ottawa during 25 – 28 February 1997, this Workshop Report was developed, discussed and agreed.

The May 1999 meeting of C3I Group directed TP-11 to add a COE Security Appendix to this Report, which is now included and issued as a 2nd Edition.

The Report 2nd Edition has also been marked UNCLASSIFIED UNLIMITED DISTRIBUTION.

Enquiries about this Report, its content and its current status, may be addressed to the Chairman or National Leaders of TP-10.

TTCP C3I TP-10 (Distributed Information Systems)

Membership Roster of Chairman and National Leaders

Chair	Mr. John Laws DERA Malvern UK	laws@dera.gov.uk
NatlLdr Australia	Dr. Iain Macleod DSTO C3 Research Center Canberra	Iain.Macleod@dsto.defence.gov.au
NatLdr Canada	Dr. John L Robinson Communications Research Center Ottawa	John.Robinson@crc.ca
NatLdr United Kingdom	Mr. John Tindle DERA Malvern	JWTindle@dera.gov.uk
NatLdr United States	Mr. Carl A. DeFranco USAF Research Lab Rome NY	defrancoc@rl.af.mil

John Laws
Chair TP-10
DERA UK
18 April 2000

Table of Contents

1. Introduction

To support the global C4I missions of the future, with their emphasis on rapid deployment, reachback, timely response and multiple mission capability, requires a seamless information environment from the sensors to the execution elements. The goal is to provide the right information, at the right time and the right place. Physically this will be implemented as a collection of interconnected information processing centers, each supporting a specific portion of the mission (e.g. planning, logistics, intelligence, monitoring, assessment, etc.). These centers will be functionally integrated to provide the perception of a single, uniform, information environment, which provides to all of the users a common, consistent and current view of needed information, even though the users and the information sources may all be physically dispersed around the globe.

There are many differing and often conflicting demands on the implementation of this C4I Information Environment. Future missions will be predominantly Joint/Coalition efforts. This raises the issue of heterogeneity at all levels of hardware, software, data, interfaces, communications, etc. In addition releasability and data interchange must be controlled consistent with the definition of the coalition activity. The explosive growth of technology requires a strategy for orderly evolution which accommodates the infusion of new technology, and the added functionality it provides, without extensive disruption to the existing functionality. The rapidly declining defense budgets mean that many application systems will be long lived, creating the necessity for accommodating both new and legacy software within the same information environment. This budget decline, as well as the fact that the military no longer represents the dominant force in the information technology arena, dictates that our future information environment be implemented primarily with commercial-off-the-shelf components, both hardware and software.

One of the predominant technology approaches being investigated for satisfying the myriad of needs of future C4I systems is the Common Operating Environment (COE). Some of the key features which make the COEs attractive are the standard core components which may be reused by multiple applications, the standard interface definitions to support interoperability, the commercial off the shelf(COTS) availability of the components, and the potential for improved scalability, survivability and evolutionary growth. To assist in assessing COEs as a potential implementation solution for military applications, S-Group tasked its subpanel on Distributed Information Systems (STP-10) to

"...review both civil and military developments of existing and projected Common Operating Environments (COE), and to assess their application within military systems. The assessment will include a determination of the boundary of a COE and to identify whether there is a common understanding and set of terminology to define COEs. Based upon the assessment of the current state of the art, the study will identify aspects relevant to future military distributed systems, and identify future research required within the TTCP community."

The following report summarizes the efforts by STP-10 to

- Baseline Commercial Technology in DCEs and COEs
- Assess Future Trends/ Capabilities in DCEs / COEs
- Analyze Representative Military Applications
- Assess Deficiencies and Needed Development

While certain deficiencies were noted in current COE capabilities, there is ample evidence that COEs will provide a significant benefit in building C4I systems with levels of functionality far exceeding what is currently available. The successful implementation of military systems based on COEs require careful consideration of many non technical factors such as acquisition strategy, operational policy and enterprise structure. The report concludes with a series of recommendations for consideration by S-Group.

2. Common Operating Environment (COE) Description

2.1 Introduction

The C4I systems of the future will , in fact, be large distributed information systems, made up of numerous processing cells performing the functions of planning, assessment, execution control and monitoring etc. The differing roles of these various C4I processing centers will levy widely varying performance demands. Those elements of the system dealing with time critical operations must accommodate all of the temporal aspects of processing, data, scheduling and prioritization. Elements dealing with mission assessment or weather support must be able to handle large complex computer models. Elements dealing with collaboration and planning must have extensive linkage to multiple DBMSs as well as a flexible and effective human computer interface. Several and possibly all of the different modes of operating may have to be accommodated within a single coalition operation. In addition the growth in types of missions will change the mix of application which must be supported. Major Regional Conflicts have differing needs from Operations Other Than War or Humanitarian Relief . What begins as one type of mission may evolve into another type, with the need for the information infrastructure to dynamically accommodate the change.

One of the current strategies for designing and implementing systems capable of meeting this broad spectrum of demands is the concept of the Common Operating Environment (COE) (Figure 2-1).

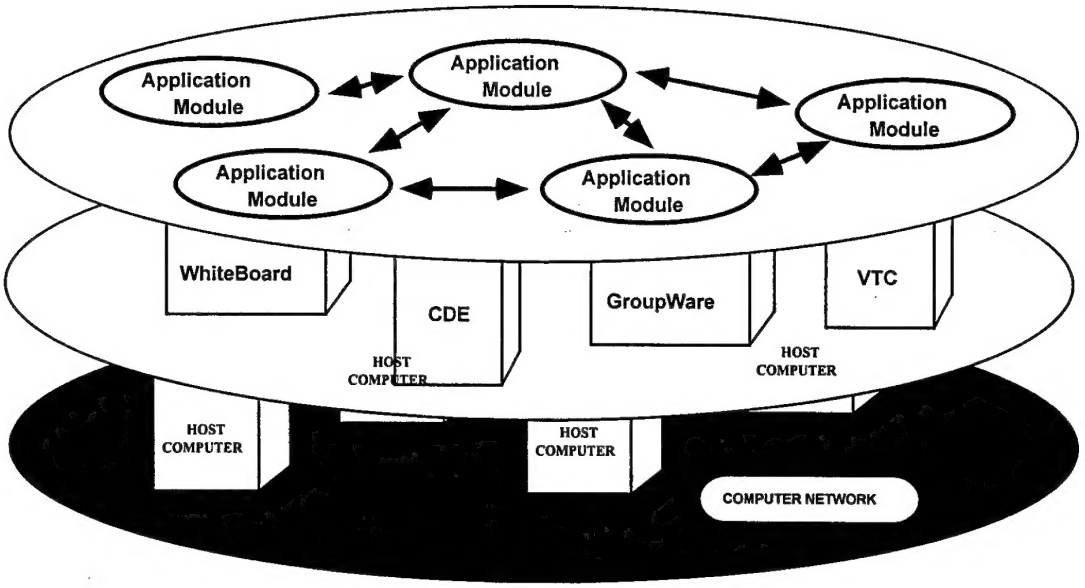

FIGURE 2-1.

Common Operating Environments

To support global operations, the COE will be built on top of a Distributed Computing Environment (DCE). The COE/DCE provides a rich environment which provides for both integration and interoperability among individual applications and tools, and also supports the concept of partitioned, distributed computing across a global information system. This section deals with the defining characteristics of the COE. A later part of the report will deal specifically with the characteristics of the underlying DCE.

A COE/DCE provides an infrastructure, described by a set of standards, which allows the easy integration of different application domain tools. When operating across a global network, it supports data interchange among the differing applications and databases which are dispersed among the globally distributed computing resources. A COE incorporates a system architecture which defines structure of the system. The COE defines an approach for implementing the architecture and incorporating domain specific applications. In the case of a large distributed set of resources the base infrastructure is implemented in the form of a DCE. The COE establishes a set of guidelines and standards which define how software elements are implemented and integrated. Ideally ,this results in a "plug-and-play" environment with consistent and well defined application programming interfaces, which from the application developer point of view provides a "common look and feel " into the system infrastructure.

To describe a COE it must be considered from several points of view (Figure 2-2.). These include the user requirements, the functional capabilities to address these requirements, the underlying physical architecture and the hardware and software elements that are used to implement the COE.

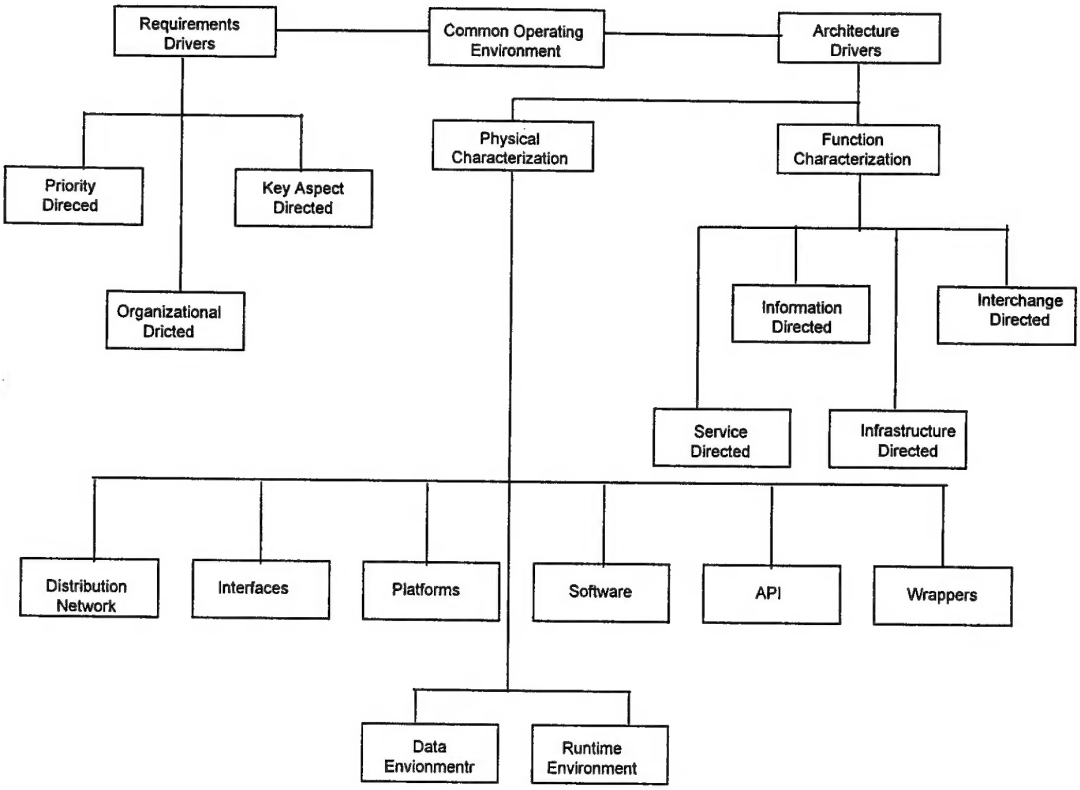

FIGURE 2-2

COE Taxonomy

2.2 How Organizational Requirements Drive COE Design

A Common Operating Environment (COE) may be conceptualized in many forms. What drives the decision making process are the prime requirements to be obtained by selecting a level of commonality and interoperability between components of the architecture. Commonality is not an end in itself. A decision for a common element across systems, functional domains or organizations can only be made on the basis of some common set of requirements.

If the chief requirement for the computing domain being addressed is, for instance, "the need for interoperability with another system" then components of the solution will be selected based on that important requirement. Where the chosen elements meet a need and are common, they may be deemed appropriate for inclusion in a decision to standardize on those components as part of both environments, the newly designed system and the system with which it is required to interoperate. If other factors weigh equally heavily in the requirements of the new system, they may affect the choice of elements that equally meet the first requirement .

"Common Operating Environment" is a relatively recent term. The Defense Information Infrastructure documentation ascribes the concept to the US military Global Command and Control System (GCCS). The elevation and need for the term probably sprang from a change in the acquisition control structure and acquisition processes away from decentralized acquisition at a time when, correspondingly, systems were becoming more decentralized. Previously, the level of compliance to broader system specification had met with limited success. However, the specification of commonality has also shown that other important user requirements sought for the systems of organizations can be met by attention to mutually common components or elements.

Despite the recent advent of the term, COEs are not in any way a new concept. Organizations with well-defined areas of information systems control have always made decisions on standardized elements of their computing infrastructure and on the common use of facilities between users units. Published systems IT architectures or the adoption of vendor-produced technical architectures, such as IBM's System Application Architecture (SAA) or Digital's Network Application Support (NAS), by organizations were examples of the adoption of elements which support Common Operating Environments. However, the advent of more strongly-formed alliances between organizations and the increasing complexity of information systems control and expenditure control within organizations has prompted definition of terms such as Common Operating Environment and the attempt by management to forge mutually advantageous standardization activities between parties who have a common interest.

There are two major drivers in the characterization of a COE. The first is user requirements that the COE is supposed to satisfy (e.g. interoperability, scalability, etc.). This provides a Requirements Driven view of the system. The second major element is the hardware and software infrastructure which is used to implement the COE. This provides an Architecture Driven view of the system.

2.2.1 Requirements Drivers for COEs

A requirements directed COE takes, as its basic mandate, the meeting of key stated requirements. The selection of common elements of the COE come from the same priorities being evident from systems which the domain of interest covers. For instance, if the domain is a coalition of libraries who need to share borrower information, interoperability of the libraries borrower systems may be an important requirement. This requirement will cause the common selection of components to meet the inter-library (cross-organizational) requirement.

2.2.2 Priority Directed

A requirements directed COE that is also priority directed has:

- a means for gathering a large number of the requirements for the system from a broad spectrum of identified interested or affected parties.
- A way of prioritizing those requirements
- A way of producing potential solution sets to meet those requirements
- A method for selecting common elements for the COE from the solution which are able to meet the most important or largest set of most important requirements

This classification makes no assumption of how the requirements are stated or ascertained. For instance, requirements may be stated from a technical solution perspective rather than from users need. A solid linkage between architectures and requirements gathering may result in the correct level of requirements specification.

2.2.3 Key Aspect Directed COE

Many COEs are selected on the basis of one or two overriding requirements. For instance, statements such as "in developing this COE, we are seeking to save money on training" are common. A single, or a small number, of overriding goals drive the decision for commonality. It is obvious to the people directing the COE effort that it would be beneficial to solve the particular problem.

One potential problem is that sometimes a solution to one problem causes another requirement to be disadvantaged. This is not a problem if the benefit of solving one problem far outweighs the disadvantage of moving further from meeting another perceived need. Without an adequate analysis and a considered focus on your priorities, as in the Priority Directed case, how can we know the impact of our decisions? For instance, if the overriding perception is to standardize elements with the specific goal of meeting cost objectives, a cost directed COE, we may choose elements that directly inhibit the ability of our system to scale easily. If we choose elements solely for their scalability, we may impinge on a goal for a manageable architecture.

2.2.4 Organizationally Directed COE

Mandatory COE specifications sometime come about because of the exercise of control by elements of the Organization. For instance, during the early stages of Local Area Networks (LAN) the explosive growth in acquisition of network technology by departments outside of the traditional central information systems management threatened to usurp their control. Specification of approved LAN environments, while helping maintain consistency, also help to re-establish the role of IS groups in the LAN acquisition process. Organizational structure can create localized requirements for control which may override the broader organizational requirements. Such exercise of control may not be for any power motivation but may merely be a response to anxiety about a shift in organizational structure.

Organizational divestiture in large corporations and government departments has allowed divisions to attain the critical mass required to effectively manage their own IT infrastructure. Depending on stewardship and motivating behavior, many attempts to enforce specific infrastructure from the top may fail without a case made for a broader set of requirements. For instance, beyond a certain size, purchasing in volume by pooling purchases may fail to realize any incremental gain.

Specification of a COE can result in significant structural shift as supporting organizational infrastructure is established to support the COE itself, such as:

- program offices
- compliance testbeds and inventories
- lobbying groups for particular outcomes and control
- packaging and roll-out release capability
- COE related outcomes directed research
- documentation and marketing literature

2.3 Architecture Drivers for a COE

The architecture for a COE can be viewed both from a functional and a physical point of view. From the functional point of view, it provides a uniform integration and execution environment for a wide variety of applications. It uses a modularized approach to define and implement both system services and applications. It can be configured to optimize against a desired set of organizational goals, to support an organizational model for interoperability, to provide a uniform environment for execution of a wide variety of applications, etc. Following are a few examples of the functional characterization of COEs.

2.3.1 Functional Characterization of a COE

2.3.1.1 Information Directed COE

An Information Directed COE is one which focuses on the standardization of information and the facilitation of exchange of information or data within the system on which we are focusing. COEs focused on information or data look for standardized data elements, data models, data management policies for structured data and common information repositories, compound document architectures or standardized document interchange mechanisms for less structured information. A key dichotomy are the COEs that are focused on standardized static storage, search, access of information or data and those which are focused on the interchange of data between systems.

2.3.1.2 Interchange Directed COE

Interchange Directed COEs focus on getting data or information between systems. Syntactic correlation of data elements is not necessary when databases are replicated. When data or information is interchanged significant attention has to be applied to understanding the level of common data and ascertaining the level of understanding between participants or systems of the meaning of that data or information. Gateways, translators or data integration tools are indicative of this environment.

Intermediate interchange formats are also a key element of this approach. These are data models developed particularly for the purpose of facilitating the interchange of information. Typically, effort is applied to other elements beyond the data model, such as the description of common classification schemes and common terminology to reduce semantic ambiguity.

2.3.1.3 Service Directed COE

A Services Directed COE has as a prime requirement the provision of services to the next level of architecture, which are a function or set of functions to be performed. The implemented architecture uses recognized standard programming interfaces to provide its

services to application functions. A set of application functions conceived to meet a users objective is the machine side of the man/machine boundary of a system.

2.3.1.4 Infrastructure Directed COE

An Infrastructure Directed approach is often associated with a Services Directed approach to a COE. In this case decisions are made on what elements of the system are required to support the functionality of a service of the system. For instance, where a users process or standard operating procedure requires the movement of information from one person to another, an electronic mail system may be considered key infrastructure, particularly if it is the only such mechanism for doing so (or at least perceived to be). This consideration may prompt the mail systems inclusion in a COE that encompasses the organizational scope of that process. Networks are typically considered part of the infrastructure and where commonality facilitates the provision of a particular key users service, they are typically standardized.

2.3.2 Physical Characterization of a COE

The physical aspects of a COE must both reflect the functional characteristics and optimize against the organizational requirements. It will be composed of a large collection of heterogeneous elements which must interoperate to provide the appearance of uniformity to the application builder. It must be evolvable to accommodate the turn over of the technology every couple of years, as well as the introduction of new elements to accommodate changes in function or scale. All of these changes must be accommodated in a seamless manner. The following paragraphs provide an illustration of some of the typical items that make up the physical instantiation of COE.

2.3.2.1 Distribution Network

Network topology affects the choice of elements and the decision to incorporate networking elements as part of the infrastructure. If all communication is on a single local area network (LAN) at a single site (an intra-LAN) then standardized components will be dictated by the underlying architecture of the environment. If the network spans multiple sites, decisions need to occur if the two LANs are of differing type. Standardized methods for signal and protocol conversion may be needed. Where a Wide Area Network is deployed, there will also be decisions on carriers to be used, protocols and connectivity options and costs to bear. Gateway approaches between WANs also have to be taken into account.

2.3.2.2 Interfaces

Standardized interfaces occur at all levels of a COE and include hardware, software, data, applications etc. These interfaces are one of the primary means of achieving

interoperability among dissimilar elements. Hardware standards have received the most attention and are the ones most people are familiar with. National and International bodies have developed industry wide standards which provide uniformity in the description, functionality and manufacture of hardware elements. For instance, there are IEEE standards for CPU architectures, Input/Output sub-systems etc. These standards do not mandate implementation characteristics but they may specify intended operational design parameters and architectural limits, tolerances etc. particularly for interfaces, which are the interaction point with other hardware or with users. The evolution of the networking technology accelerated the development of data interchange standards to allow dissimilar computers connected to the network to efficiently exchange data. The area of standards that has been most effected by the evolution of COEs are the Application Programming Interfaces (API) which will be discussed under the Software part of the physical architecture.

2.3.2.3 Platforms

Platform or component specification is by far the most common form of hardware specification in a COE. The extent of specification varies considerably from broad system package specification down to the firmware update or the individual fixes in the field. A given version of the hardware may be specified. It would be also possible to specify a given source for a particular element, for example, a graphics board produced by a particular company when there are several producers making the same component under license.

2.3.2.3.1 Hardware Dependencies

Dependent elements are an important consideration when developing a common operating environment, for instance:

- chosen 3D graphics software may only work with certain graphics adapters,
- printers may only come with a serial interface,
- token ring network cards only work on token ring networks and
- computer motherboards are engineered for specific chip placement.

However, dependencies can cause considerable inflexibility in configuration. For instance, a recent civil aviation systems acquisition specified a mandatory requirement that a given circuit board level hardware component have a revision level for the firmware on the board that was identical in all systems acquired and that spare parts be available for the board at exactly the same revision level for ten years. Some Multi-level secure operating systems require particular processor releases and system configurations. Certainly specific hardware or central processing unit architectures (CPU) and specific operating systems are commonly part of many COEs.

2.3.2.4 Software

The software which comprises the COE as well as the application software that uses the COE is of many different types, from many different sources. It is developed by different groups, for different purposes, using different development environments, different

languages, different levels of design rigor, etc. The goal of the COE is to support integration and interoperability across this complex environment. The key concept to providing this capability is the use of open system standards. These standards concentrate on the definition of the input/output and interface characteristics of the software modules. If the software modules conform to these standards, then they will be reusable within any COE which uses the same interface standards. The internal implementation of the modules is not a concern since it is masked from the other modules of the system.

There are many classes of functional software which make up or use the COE. These include the infrastructure software, the common services software, the common application support software and the application software. An increasing majority of the software packages will be commercial-off-the-shelf. To the extent that they conform to open system commercial standards, the easier will be their integration into the COE. The same will be true for the specially developed software. The majority of the specially developed software will be at the application level. The efficient integration of the applications is dependent not only on adherence to the API, but also on the effective use of the common application services provided within the COE.

From a different point of view, the software can be divided into two groups, software which has previously been developed and at some later time is incorporated into the COE (legacy software) and new software which is developed specifically for inclusion into the COE. Legacy software often has a "software wrapper" written for it to encapsulate all input/ output and interfaces and make it "appear" compliant with the COE specifications. While this may result in performance penalties, it still allows the legacy software to operate within the COE as a reusable module. Newly developed software is designed to the COE specifications and standards to both be compliant with architecture as well as take advantage of common services that it provides.

Before being included in a COE software modules must undergo a series of compliance tests. This activity, which is supported by a set of compliance tools which are themselves part of the COE, can verify conformance to the COE standards. To accommodate the wide variety of software elements, and the multiplicity of ways that a module can be made to conform, multiple levels of conformance are normally defined. This can control the extent to which the integrated module can utilize the COE shared functions. Integration into the COE implies that the element works correctly within the COE runtime environment; does not adversely interact with other elements; conform to the COE standards; has been validated by the COE tools; and can be installed within or on top of the COE by the COE installation tools.

2.3.2.5 Application Programming Interface

The API is the portion of the COE which presents the system "look and feel" as viewed by the application. This is what is often used to describe what the COE is to a potential user. It is COE specific since it presents to the application programmer the view of the underlying COE services. It is comprised of a series of standards which support the logical binding of application programs to the services provided by the COE. The set of

standards must be rich enough to allow for the easy integration of a wide variety of application types, as well as make full utilization of the all of the services of the COE.

Many of the API's will be commercial products (e.g. X-Windows, MOTIF) while others will be COE domain specific such as a DBMS interface or a data (e.g. map) server interface. For the development of new software, the API can provide a capability that makes the task easier. It can provide standardized ways of accomplishing repetitive tasks such as building windows based user interfaces. It can also provide efficient coupling to COE services such as common data servers, interprocess communications servers or utility support servers. It provides a commonality and unifying theme among the applications which also aids in the integration of the large set of applications that make up, for example, a large command and control system.

The API is the mechanism through which legacy software is also integrated into the system. The task can be simple or very difficult, depending on the degree of difference between the standards which define the API and the mechanisms used in the legacy software to define external interfaces with other programs, definition of user interfaces, and the methods used to create bindings between the application program and its datasets. The external interface issue can often be handled by software wrappers which intercept all input/output and make the necessary translations to be compatible with the standards of the COE. The user interface and the dataset bindings can create more of a problem. These could in fact result in a major rewrite of the application. For this reason, multiple levels of compliance or compatibility are usually defined to indicate the extent to which an element conforms to all of the strategies, architecture and standards of the COE.

Compliance can be at a superficial level represented by "interface" to the COE through relatively unstructured data sharing and some sharing of common standards. Intermediate levels of compliance deal with the issues of structured data sharing, common user interfaces and interoperability. Full compliance includes the modularization as defined by the architecture, adherence to the software style guide, interfaces only through published API's, full use of common services, etc. As legacy software is replaced in an evolutionary manner by software generated specifically for use in a COE the level of compliance will naturally rise.

2.3.2.6 Wrapper/Adapter

Wrapper architectures are designed to facilitate access to legacy data repositories that may have been part of older systems environments but whose data still provide valuable information to the current operations.

Wrappers provide a bridge between legacy software and the COE to allow migration to new architectures and software in a timeframe the organization may better be able to handle. COEs may provide for the possibility to migrate over time under the controlled auspices of a migration plan.

2.3.2.7 Data Environment

Concurrent with a decision to share data models come implementation considerations. Modern database technology allows data to exist in separate locations and to be seen as one logical entity. Replication to remote sites can also be automatically managed. The structure of the data distribution mechanisms will impact the selection of other COE elements. Knowledge of where information is stored will have dependency effects on the ability of the topology of the network to be fault tolerant. The distribution of key data across the organization may well drive the decision to make the network part of the COE as key infrastructure and add more resources to the management of supporting elements of the COE. If replication occurs without a solid protocol between parties for the exchange then corruption may occur. Event-driven updates (often called database triggers) may prompt updates in other systems. The decision to engage in data or information "warehousing", a term for making a decision to organize your data repositories a certain way to fit particular users needs, can quite possibly drive the nature of the database tools chosen by the COE developer.

Mandating a shared data model, or trying to rationalize the outputs of another functionally similar data model can cause considerable difficulty. Considerable effort has been expended on attempting to rationalize the data element inconsistencies between some supposedly equivalent models. This was discussed more fully in the section on Information Directed COEs.

When dealing with data sharing, particularly in situations where a multi-database integration is being performed, it is important to understand, at any given time, which logical view you have of the information you are working with. This will affect confidence level when data or information is being semantically correlated. COE developers may impose particular restrictions on database activities to control issues of integrity and quality of data.

2.3.2.8 Run Time Environment

The implementation of an operational COE can involve the specification, design, integration, test, and execution of hundreds of software modules. Each of these must function in conjunction with some subset of other modules, and not adversely interact with any others. Configuring an operating COE is a step by step process which begins with the basic operating system elements, upon which are layered the COE service modules, which serve as a base for the applications through the application programming interface. Dataflows and process bindings must be established and maintained among the interacting elements. Directories, file systems, databases, access control lists, process identifications etc. must all be established and maintained.

The run time environment provides the configuration and execution support for both the service modules that make up the COE and the application programs that execute within the COE. It incorporates the functions to generate and incorporate service and application modules into the COE. The specific elements of the run time environment are dependent on the particular host computers included, the number and type of services provided by the COE,

the mix of COTS/GOTS and specially developed software which makes up the COE services, and the domain specific attributes of the application.

The run time environment has aspects which are both system specific and domain specific. If the COE is configured across a heterogeneous collection of computing elements then the run time environment will also be heterogeneous. Heterogeneity at the operating system level will be masked by the COE/DCE. Heterogeneity at the application support level will be masked from the application programmer by the API or the servers. Other aspects, such as real time scheduling or thread management , must be visible to the application. Normally, a set of automated tools are part of the COE runtime repository to assist in the development and installation of the elements of the COE.

2.4 Implementation Considerations for COEs

2.4.1 Software Configuration Management

The COE is composed of very large quantities of software independently developed by different groups, potentially over a long period of time. Even though they are guided by a set of specifications, the development of a COE is plagued by many of the common software engineering problems of any large software system. Typical problems include version control, software trouble reporting and solution, specification interpretation, specification evolution, conformance testing, release control etc. To keep control of these problems demands a comprehensive Configuration Management System (CMS).

A central software repository, from which version controlled release of software modules is accomplished, establishes a baseline for any configuration. Before a module is accepted into the repository, it is subjected to a set of conformance tests to assure that it meets the COE specifications. After each "bug" fix or version upgrade the module is retested for conformance. In addition to individual module tests, interoperability testing must also be accomplished before modules are certified for release as part of the COE. Standard configuration modules are provided from the repository to any user. A documentation repository must be maintained which is consistent with the software repository and provides current documentation on all elements in the repository.

An on-line Software Problem Reporting (SPR) system must be maintained to both identify and track the correction of software bugs. This should be a "user friendly" system which accommodates the fact that users, developers, administrators etc. have different views of the system and often differing description of bugs (or undocumented features).

One of the primary attributes of a COE strategy is incremental development and evolution. The "build a little, test a little" philosophy is fundamental to the COE. Incremental development with version controlled releases and adequate pre-release documentation should be an integral part of the CMS

Separation must be maintained between developer configurations and operational user configurations. The developer configurations will include untested and undocumented software which has not been subjected to conformance testing and version control. Prior to developmental software inclusion into the central software repository for distribution to users it must undergo the full set of conformance and interoperability tests.

2.4.2 Software Extensibility

Much commercial-off-the-shelf software (COTS) and some custom software provide some mechanism to enhance the functional quality of the software to meet needs. An example is the detailed macro language provided in any spreadsheet today. Some application environments have highly tailorable tools, either by an application programming interface or by a macro language.

The decision for a particular software product, especially for COTS, is often made on several assumptions:

- that using the same software product will result in ease of integration with other tools
- that learning/start up time will be minimized in moving across environments due to common applications
- that upgrades will be easier
- taking advantage of commercial development investment

Applications which are highly tailored or customized beyond the limits envisaged for the provided customization capabilities can actually result in difficult integration of two applications even when they are running the same software, added complexity due to non-familiar interactions due to tailoring. Upgrades of software make assumptions about the degree of tailoring. Significant tailoring beyond that amount may result in significant porting of software to the new version.

A COE developer may decide to do any of the following:

- enforce a process of ensuring that excessive tailoring does not occur
- remove tailoring completely, resulting in an out of the box (OOTB) application
- ensure that the acquisitions `process acquires tools that support a great deal of extensibility, certainly with a design space beyond the envisaged level of customization.

2.4.3 Generic Capability List for DCEs/COEs

There have been a number of COEs defined, each with different drivers upon which they are attempting to standardize. The table below gives an extensive, generic list of categories, with individual COEs consisting of a subset of elements

Category	Items	Comments
Hardware	Platforms Hardware/Firmware Revisions	
Software Engineering	Interface Definition Language (IDL) Version Control Programming Language Development Environment	Covers tools and standards for system/software life-cycle processes.
User Interface	Look and feel Windowing system APIs Desktop Managers	
Data Management	Database Services, including: Replication Remote Data Access Query Language Database Management Dictionary Services Directory Services	
Data Interchange	Document Interchange Formats: OA formats Hypertext Multi-media Character sets and alphabets Business services: EDI Messaging Encoding Graphics/image formats Audio data Data Compression Real-time data services	Mainly covers data formats, but includes higher level protocols for data exchange. Includes military messaging as well as EMail For example: UUENCODE Including still and moving images Includes video/audio services and 'ticker-tape' style services
Network Services	Communication Protocols LAN/WAN Name Services Addressing	 Includes wire protocols and interconnection E.g. DNS or X.500

Category	Items	Comments
Operating System Services	OS API	
Security Services	Authentication Authorization Confidentiality Integrity Non-repudiation Key Management and Distribution Accounting and Auditing Security Domain mediation	Includes all the usual candidates for security services Allows the exchange of information between domains at different classifications.
Internationalization	Character Sets Data Representations	Standards and conventions to facilitate the re-use of software or systems within different nations.
System/Network Management	System Management Network Management User Management Fault Management Disaster Recovery	All of these are examples of issues regarding system life-cycle and its management. This obviously also ties in strongly with managing the security issues of a DCE/COE.
Distributed Computing	Distributed Database Management RPC Distributed file/printing services Remote GUI presentation Name Services Time synchronization Transaction processing Object Services Distributed System Management Load Balancing Componentware	 e.g. X11's remote display capability Includes location and invocation of object based services E.g. JAVA, ActiveX
Application Services	Messaging Bulletin Boards Groupware Workflow	 Automated data transfer
Alert Services		Designed to support the rapid and widespread distribution of information.

Table 2-1

Generic Capability List for DCEs/COEs

3. Description of Representative DCEs

3.1 Open Software Foundation Distributed Computing Environment (OSF/DCE)[1]

3.1.1 Overview of the OSF/DCE

3.1.1.1 The history of the OSF/DCE

Distributed computing is not new. Several thousands of sites around the world would claim to be running distributed computing services. However, most of these implementations are based on proprietary solutions, and only offer partial interoperability solutions, relying on the expertise of developers to patch systems together.

The main challenges of any distributed system are:

- Interoperability, the ability of two systems to exchange information in a standard form. This may extend to users of one platform running programs on another platform and even applications being distributed across multiple computers.
- Transparency, that is the user can use any local or remote system and also be unaware of whether the source of data is local or remote.
- Consistent support for heterogeneous computers from multiple vendors.
- Identification of distributed services and resources.
- Provision of good security.
- Support for end users, application developers and system administrators.
- High availability and good performance.

OSF DCE is an interoperability standard for open distributed computing. It was developed by the Open Software Foundation (OSF). OSF is a not-for-profit organization that uses consensus to create standards for various vendor independent concepts. It is a consortium of over 360 members including commercial, government, and university groups. OSF was originally a technology integrator and distributor of Open Systems Technology with over 300 employees worldwide. Its technology products have included an operating system (OSF/1) , a visual user interface and toolkit (Motif) and a distributed computing environment (DCE). Today, OSF has merged with X0pen to form Open Group and no longer develops products. Its main role is to oversee the development of OSF products by industry.

OSF issued its request for DCE technology in 1989. A year later, after several submissions and reviews, an OSF DCE technology was selected. It then took another two years for OSF to deliver version 1.0 of its DCE product. The first OSF DCE vendor

[1] "Distributed Computing Environment:An Architecture for Supporting Change" J Mansfield and J Clothier DSTO Report DSTO-CR-0007

products began to emerge in late 1992 to mid 1993, but these versions suffered teething problems and reliable OSF DCE implementations were not widely available until late 1993 to early 1994.

3.1.1.2 Basic Services Needed By Distributed Systems.

Computer programs or rather applications which execute on the same machine are known as centrally run programs. In the 1970's and 1980's most programs were like this. The program was executed on a central mainframe computer. A number of terminals were attached to the mainframe and these acted as simple display devices which carried out no processing themselves.

Centrally executing mainframe programs are very efficient. Control overheads are low when management of a program is centrally administered. However, organizations that rely on such computer services are vulnerable because if the central computer fails, all operations are lost. One way of overcoming the vulnerability of centrally executing mainframe programs is to run a duplicate machine. But mainframe computers are expensive and there is no guarantee that both systems will not fail at the same time.

The alternative to central execution is distributed execution. Distributed execution is when the program or application is executed using a number of different computers. In these cases the application must make a request to another computer for a service. The recipient computer then executes the requested service and sends the response back to the requesting computer. Computers requesting a service are known as clients. Computers providing a service are known as servers. It is possible for a computer to act as both a client and a server (Figure 3-1).

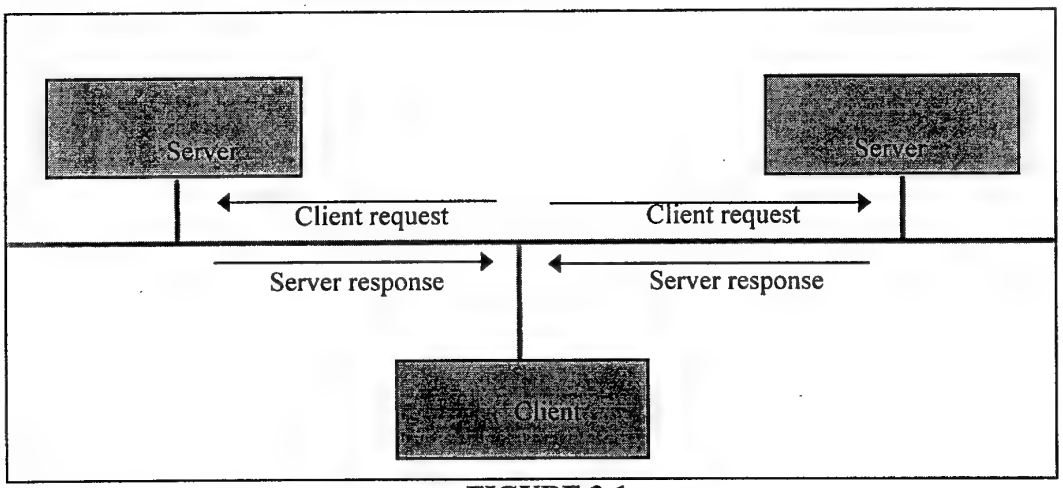

FIGURE 3-1

A very simple client/server architecture.

(A single application uses the services of several computers. This requires the part of the application hosted by the client computer to request services from the another part of the application hosted on the server computer.)

As soon as an application becomes distributed then there are a number of services which become critical to its smooth execution.. Some of these services are necessary to give the illusion that the application is running on a single machine, others such as time and security servers perform functions that are only necessary because the application is running on several machines. For administrative and operational purposes distributed services are collated into a cell; i.e. a group of users, systems and resources that typically have a common purpose and share common services. At a minimum, a cell consists of a directory (name) service, a security service and a time service. Usually a cell consists of nodes in a common geographic area, but geography does not necessarily determine its boundaries. Boundaries of a cell, in terms of the number of systems and users, are influenced by four basic considerations: purpose, administration, security and overhead. Figure 3-2 provides an illustration of such a cell.

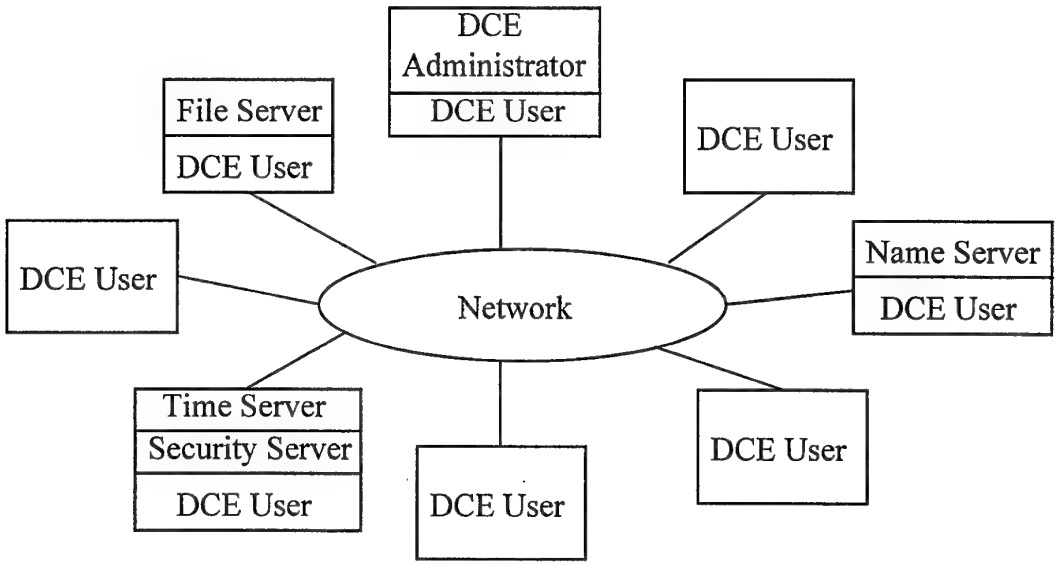

FIGURE 3-2

A DCE Cell

(A OSF/DCE cell is an independent set of clients and servers, managed as a group. Cells can be combined to form multi-cell systems.)

These services are combined into the OSF/DCE architecture shown in Figure 3-3 but in addition DCE provides an interface definition language to provide the "glue" between the client and the server.

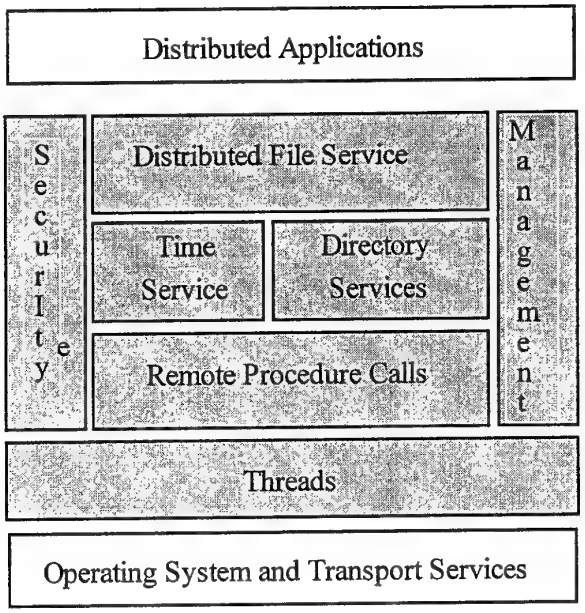

FIGURE 3-3

The OSF/DCE Architecture

(The services provided by OSF's DCE are represented by shaded boxes. They include: a Distributed File Service, a Time Service, Directory Services, Remote Procedure Calls, Threads, Security and Management.)

3.1.1.3 An Interface Definition Language.

The interface definition language allows the interface to each server to be defined in a standard way so that any client may use its services. A programmer writes the interface definition file using the interface definition language. An interface definition language compiler produces header files that support the data types in the interface definition and generates the client and server stub files. A universal unique identifier (UUID) is used to distinguish the interface. A UUID is produced using a UUID generator utility (Figure 3-4).

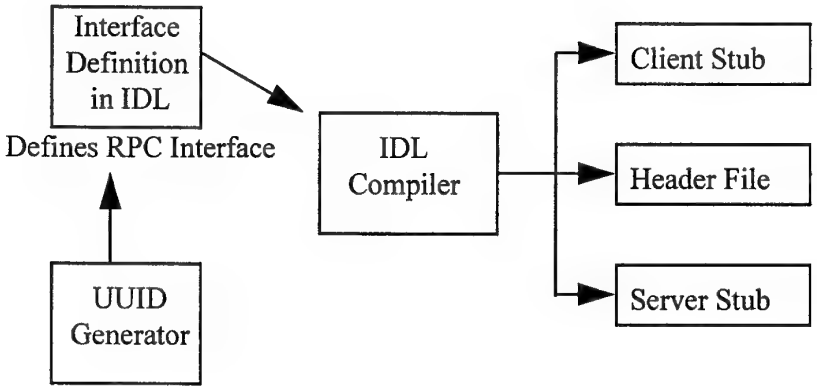

FIGURE 3-4

The UUID Generator and the IDL Compiler
(Defining the interface to each server requires a unique identifier to be generated.)

3.1.1.4 A Distributed Time Service.

In a distributed computing environment it is important that the computer clocks are synchronized, otherwise an application may show unpredictable behavior. Without this synchronism there could be chaos as, locally, each machine may have a different system time and some machines may even be in different time zones. For the purposes of file updating, transaction processing, security, backing up, compilation, etc. it is crucial that all machines have the same reference time. OSF/DCE provides a distributed time service to synchronize the time on the machines within a cell.

3.1.1.5 Remote Procedure Calls.

Procedure calls take on new meaning in a distributed computing environment. A computer program normally consists of a set of procedures which execute sequentially. A procedure may request that a file be opened and data read from it. However, it is always assumed that the file and the procedure for opening it exist on the same machine. When the file and the procedure for opening that file exist on another (remote) machine, then the requesting program must make a call to that remote procedure. This is known as a remote procedure call and involves knowledge of the communication systems used to transmit the data. Remote procedure calls permit the application programmer to use functions from separately compiled programs, that may be running on geographically dispersed computers. These procedures are called in the same manner as procedures local to the program and OSF/DCE handles all the communication problems transparently to the programmer. The local program is called the client and the remote program is called

a server. A server may access a database or carry out a complex parallel computation but all the application programmer needs to know is the name of the procedure and its parameters (Figure 3-5).

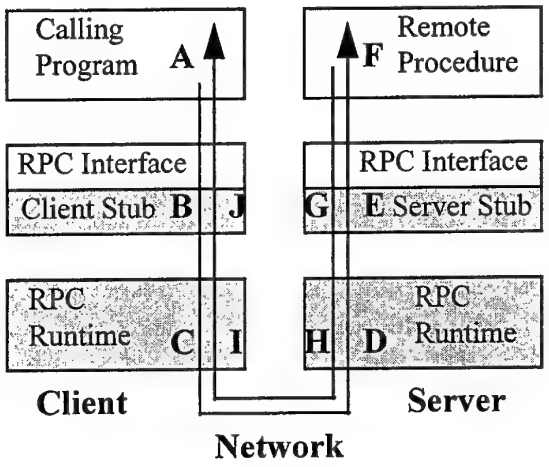

FIGURE 3-5

The RPC Path

(The path taken by a remote procedure call from the calling program to the remote procedure and back again.)

3.1.1.6 Threads.

A program which makes procedure calls to other computers obviously has the potential to slow down the rate at which an application runs whilst it waits for a response from the other computer. Threads are a popular way of improving application performance. They introduce parallelism and have been used in various versions of UNIX and OS/2 for some years. A procedure may be duplicated within the program to form threads. These threads execute independently but share static and external data. A server program using threads could handle multiple client requests simultaneously and thus minimize delays within the system. The version of threads adopted by OSF/DCE is that from the POSIX standard (Figure 3-6).

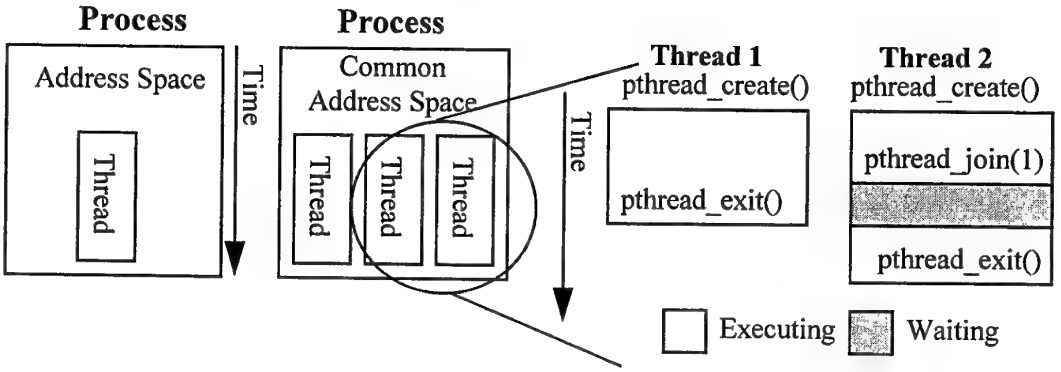

Figure 3-6

Threads
(Threads permit servers to respond to multiple clients.)

3.1.1.7 A Local Cell Directory Service.

OSF/DCE connects together a logical group of machines into a cell. The machines in a cell may be few or many, they can all be local or they can be spread around the world. However, all machines in a cell share the same file system, so all the users of the machines in the cell can see the same files. Of course, if the file the user wants is on a machine on the other side of the world it will take longer to access it than if it were local but the user need never know where it physically resides. This access is governed only by the user's authorized access rights. (see 3.1.1.11)

3.1.1.8 Distributed File Service.

Different computers have different file systems. For example, an MSDOS file system is very different to a UNIX file system. To overcome this problem in a distributed environment there is a OSF/DCE file service (DFS) that gives the appearance of a single file system which incorporates all the files available within the system. So users and computers can use this service to easily share files and the information in those files.

3.1.1.9 Access To The Global Computing Environment.

Whilst most of a user's work will be done within the cell it is natural that occasionally there will be the need to access machines outside the cell. A user can reach any machine outside the cell via the X.500 directory standard or via the Internet Domain

Name Service. If the required machine is in another OSF/DCE cell the user can become a principal within that cell with whatever access rights are granted by the cell administrator. A Global Directory Agent (GDA) makes cell interaction possible. When a cell directory service determines that a name is not in its own cell, it passes the name to the GDA (Figure 3-7). The GDA searches the appropriate global naming environment for more information about the name.

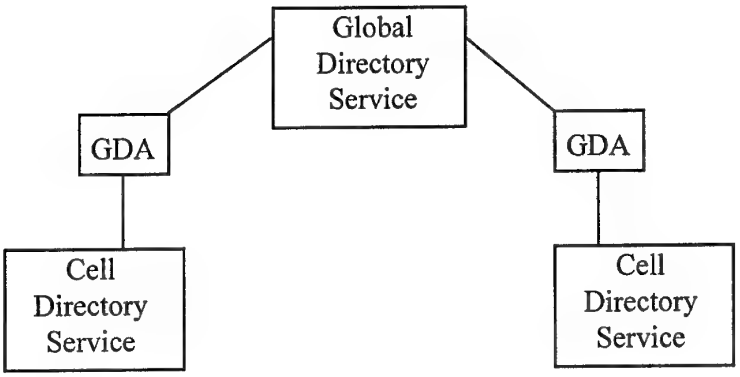

FIGURE 3-7

The Global Directory Agent
(Access to machines outside of a cell is gained through a global directory agent (GDA).)

3.1.1.10 Security.

Protection of computer resources, such as files and applications, from unauthorized access requires a user to be regarded as authentic and to have the necessary authority. This usually requires a user to enter a password as proof that he or she is who they claim to be. Access to files is then controlled through permissions or privileges associated with each resource. When the number of users is small, a single computer may manage all of the passwords and permission functions. For a distributed computer system, there may be a very large number of users accessing an even larger number of resources. It would be impractical to maintain every user's security information on every computer in the system. The answer lies in the provision of a single centralized database which serves security information. The security service uses an enhanced form of Kerberos to authenticate a user at login and when they access restricted programs. It also gives six levels of access control to partition data and databases.

3.1.1.11 Management.

A distributed computer system has a much higher management and administration overhead than a centralized computer system. Tools for administrative support which automate some of these routine tasks help to reduce the management burden of distributed systems. These tools also permit the administration of a widely distributed systems to be managed centrally, in particular the management of user accounts and data access can be governed by a single administrator.

3.1.2 Current developments in OSF/DCE

The latest version of OSF/DCE is Version 1.2.2 which was available from January 1997. This version includes major security enhancements such as public key support, improved scalability, up to 100,000 users, interoperability between OSF/DCE Kerberos and Kerberos V5 and the facility to form global groups so that a user registered in one cell can participate in another cell without needing to first be registered as a user in that cell.

It also supports DFS gateways to Novell Netware and NFS and C++ constructs in the IDL compiler.

Previously, use of OSF/DCE was somewhat expensive with a set of servers costing in the region of $15,000. Version 1.2.2 adopts an Internet style for licensing terms, i.e. clients will be free and servers will only cost several hundred dollars. A $100K unlimited, no-royalty license for OSF/DCE client binaries and a free OSF/DCE source license for internal use will be available via the Web. This licensing philosophy reflects a complete change of heart in marketing OSF/DCE and should promote a much wider use of OSF/DCE.

3.1.3 Future developments of OSF/DCE

The Open Group predict that the next major release of OSF/DCE will integrate several OSF/DCE-related technologies that are being developed as separate projects:

- ActiveX/DCOM will move to become based on a OSF/DCE substrata including DFS, OSF/DCE
- RPCs and OSF/DCE security.
- Multi-crypto RPC, public key, key escrow
- Lightweight Directory Access Protocol (LDAP) based directory services
- COM/CORBA/DCE interoperability

3.2 CORBA

3.2.1 Overview of OMG Object Management Architecture

3.2.1.1 Why Objects

Objects are abstract entities that can represent anything from data and devices to whole applications or databases. The power of object modeling comes from the requirement to define an interface around the object. This interface mediates all access to the object and provides a protection boundary around the object based on what was declared visible through the interface. The interface defines the data structures contained within the object and the operations (or methods) which are used to externally manipulate the object. Work is accomplished by invoking an operation. The invocation process is the process of using an object's operation to access an object. So through object modeling, we have a method for describing and manipulating software using standard interfaces and protection boundaries.

When objects become distributed across multiple platforms, the client code which calls the object is separated from the implementation (or instantiation) of the object. Now, an object delivery system must exist to allow and aid in the process of getting the object operation request from the client to the server which implements the object. The Object Management Group (OMG) created a series of specifications which define such a delivery system.

3.2.1.2 The Object Management Group (OMG)

The Object Management Group is a non-profit consortium of members that promotes object technology. It was founded in 1989 with the sponsorship of about eight companies, and has grown to approximately 700 members at varying levels of sponsorship. OMG strives to create a component-based marketplace through standardization. It establishes industry guidelines and specifications which range from describing the underlying support environment to providing tools and services for application development. The summation of this effort is the Object Management Architecture (OMA). OMA is the framework for an object system which is supported by detailed interface specifications. The specifications are created and adopted by OMG members, but, OMG does not implement them. The implementation of the OMA is left open to any group that wishes to use them (vendors, end-users, developers, etc.).

The OMA has a reference model which breaks the object environment into three broad areas: system oriented, application oriented, and market oriented (Figure 3-8). The system oriented piece of the reference model defines both the delivery or conveyance system and those services which are common and necessary to all objects. The

application oriented portion of the reference model groups and specifies tools and facilities which assist in the application development or execution environment. The final area, market oriented, surmises that there are sets of objects and services which are not common to all applications (in general) but are common to a vertical segment, cross-section, or domain of applications.

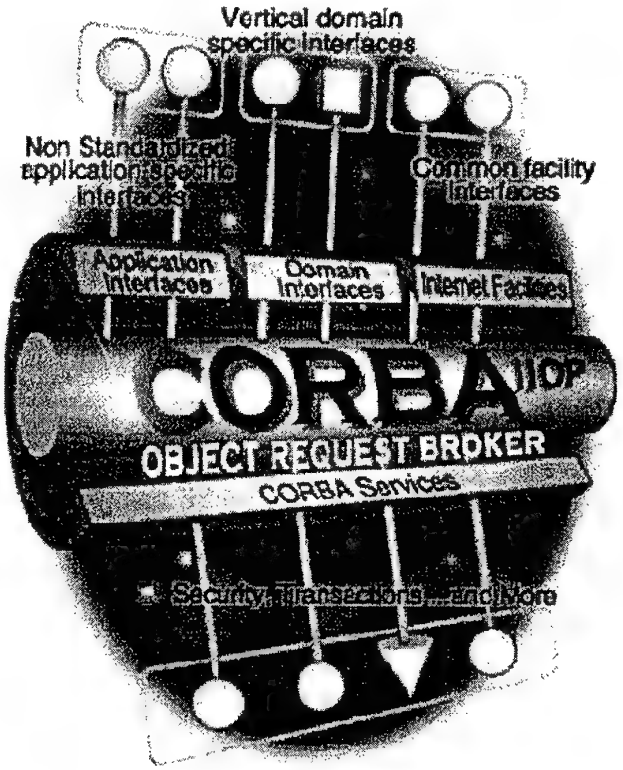

FIGURE 3-8

OMG Object Management Architecture

CORBA, the Common Object Request Broker Architecture[2], predates the OMA vision. Within the OMA, the term CORBA can best be described as a wrapper around the object request broker, object services, and common facility portions of the OMA. Thus, we have three parts of CORBA: the base CORBA-IIOP, CORBA services, and CORBA facilities. (See Appendix 9.1 for a description of CORBA Services and Facilities)

[2] "CORBA - A Guide to Common Object Request Broker Architecture", Ron Ben-Natan, McGraw Hill.

3.2.1.3 The Common Object Request Broker Architecture (CORBA)

The CORBA specification defines the client-server middleware portion of the OMA. It provides an infrastructure for object conversing, platform independence, and object implementation techniques. In general, it is a programming paradigm for creating object-based to object-oriented, distributed applications. CORBA is independent of all of these issues: vendor implementation, language (within reason), host or platform, and operating system. It comprises an interface definition language (IDL), an object request broker, a set of language programming interfaces, a set of object services, and a set of common application facilities (Figure 3-9).

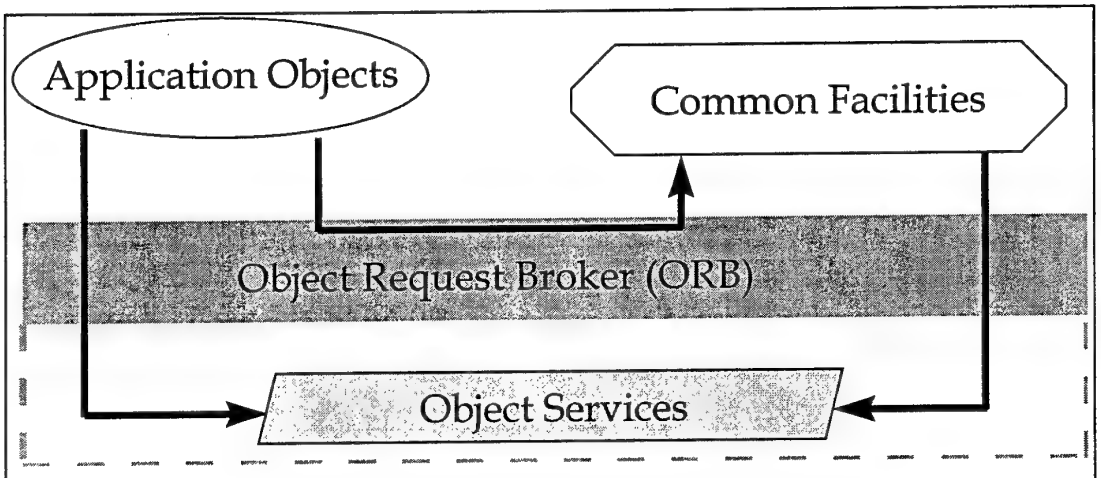

FIGURE 3-9

CORBA

(The Common Object Request Broker Architecture (CORBA) illustrates that application objects interact with both common facilities and object services through the ORB. Also, the implementations of the common facilities may also use object services.)

Through CORBA, the programmer or end-user sees interfaces to objects and services through a language sensitive, context-based lens. In developing applications, the programmer uses the IDL to define both internal and external access to objects, uses IDL mapping tools to operate within an appropriate programming language, and uses filler code (and other object/service interfaces) to "flesh out" the application requirements. The breadth of the CORBA specification can be broken into three broad areas: core, interoperability, and interworking.

3.2.2 CORBA Core

3.2.2.1 CORBA Object Request Brokers (ORBs)

The Object Request Broker (ORB) is the infrastructure responsible for the following mechanisms: location, preparation, and communication. Location mechanisms pair requests for objects with object implementations. Preparation mechanisms allow the transmission and/or reception of object requests by both the client and the object implementation. Communication mechanisms provide the base layers and primitives to deliver requests from one location to another. The ORB can be implemented in a number of ways, so it is not (necessarily) true that every node must have an ORB. At one end of the spectrum, an ORB could be a monolithic process on every distributed node of a network. At the other end of the spectrum, an ORB could be a set of function calls within the address space of a large multiprocessor. ORBs are best viewed as the environment "glue" which connects the software components within CORBA.

3.2.2.2 CORBA Interface Definition Language (IDL)

The CORBA IDL provides a way to describe the interface which surrounds an object. It has the same lexical rules as C++, but, there are extensions which allow for concepts like distribution, access control, security, et cetera. The formal grammar for CORBA IDL is a subset of ANSI C++ (Example 1).

```
interface Inventory [inheritance-options]
{
        struct INVobject
        { unsigned long   Count;
          string          Description; };

        exception NOT_ENOUGH_STOCK
        { unsigned long CurrentStock; };

        Inventory Create( in unsigned long  NumberOfItems,
                          in string         NameOfItems );

        unsigned long RemoveFromStock( in short NumberOfItems )
                raises( NOT_ENOUGH_STOCK );

        unsigned long AddToStock( in short NumberOfItems );
};
```

Example 1: A sample object definition using IDL

3.2.2.3 CORBA Language Mappings

Language mapping within CORBA provides a methodology for translating the abstract IDL into an implementation language that is used by the programmer to create an object instance. Vendors who use the CORBA standards must implement at least one of the language mappings. There are currently four languages which are supported: ANSI C, ANSI C++, SmallTalk, and Ada. The COBOL language mappings are still under development. Language mapping tools "compile" the IDL into two halves: client (operation invocation) stubs and server (operation processing) skeletons. These form the base upon which a programmer composes the object using whatever implementation

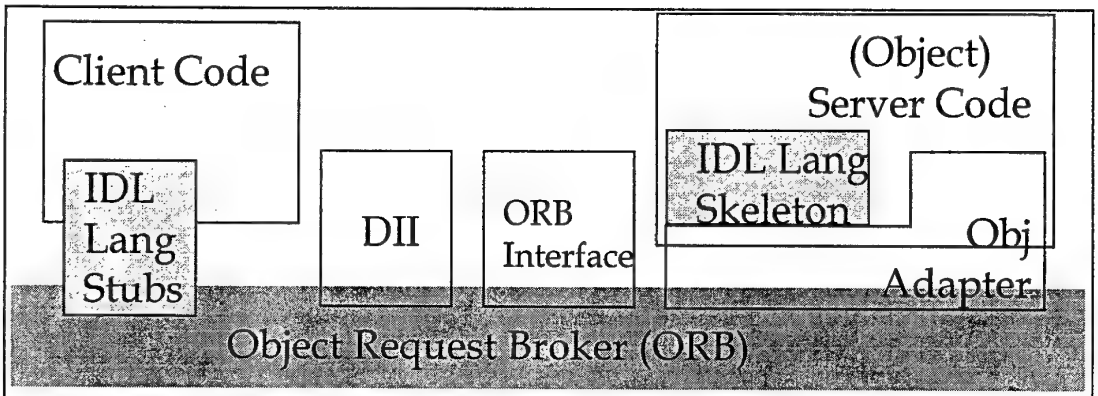

FIGURE 3-10 Language mapping

(Both clients and object implementations can make use of the "stub" routines provided by the language mapping tools to provide a function call interface to object requests.)

language is best suited (Figure 3-10).

3.2.2.4 CORBA Interface Repository

The interface repository provides run-time storage for IDL information. This information is generic, allowing it to span different ORB implementations. As a resource, this repository offers an on-line source of information that completely describes the interface to an object. The repository requires persistence and assumes the existence of a "repository object" for each object interface.

3.2.2.5 CORBA Implementation Repository

The implementation repository provides run-time storage for information regarding the location and activation of objects. The information contained within the repository is ORB specific and is intended, primarily, for use by the ORB. Ordinarily, installation
of implementations and control of policies, related to the activation and execution of object implementations, is done through operations on the Implementation Repository. In addition to its role in the functioning of the ORB, the Implementation Repository is a common place to store additional information associated with implementations of ORB objects. For example, debugging information, administrative control, resource allocation, security, etc., might be associated with the Implementation Repository.

3.2.2.6 CORBA ORB Interfaces

The ORB interfaces are a series of functions which have been defined to be common to all ORB implementations. Currently, they constitute object reference to string conversions, object reference duplication and elimination, null object reference generation and testing, and dynamic invocation/skeleton interface support.

3.2.2.7 CORBA Object Adapters

An object adapter is the primary interface used by developers to access ORB functions on the object implementation (server) side. All ORBs must implement the functions specified by the Basic Object Adapter, so programmers can consistently rely upon a set of adapter functions which are transportable across all ORB implementations. Beyond the basic adapter, ORB implementors are allowed to provide any number of alternate adapters which may be specialized for particular applications or functions.

The Basic Object Adapter provides a base level of functionality: generation and interpretation of object references, authentication and access control capabilities, object implementation activation and deactivation, object activation and deactivation, and method invocation through skeleton code.

3.2.2.8 Dynamic Invocation Interface (DII)

The Dynamic Invocation Interface allows both the creation and invocation of object requests at run-time versus compile-time. The interface defines tools for low level functionality: message creation and destruction, parameter marshaling, message invocation and reception, synchronous and asynchronous requesting, and context property support.

3.2.2.9 Dynamic Skeleton Interface (DSI)

The Dynamic Skeleton Interface (DSI) is a way to deliver requests from an ORB to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. This contrasts with the type-specific, IDL-based and language-mapped skeletons, but serves the same architectural role. The DSI is the server side's analogue to the client side's Dynamic Invocation Interface. Just as the implementation of an object cannot distinguish whether its client is using type-specific stubs or the DII, the client who invokes an object cannot determine whether the implementation is using a type-specific skeleton or the DSI to connect the implementation to the ORB (Figure 3-11).

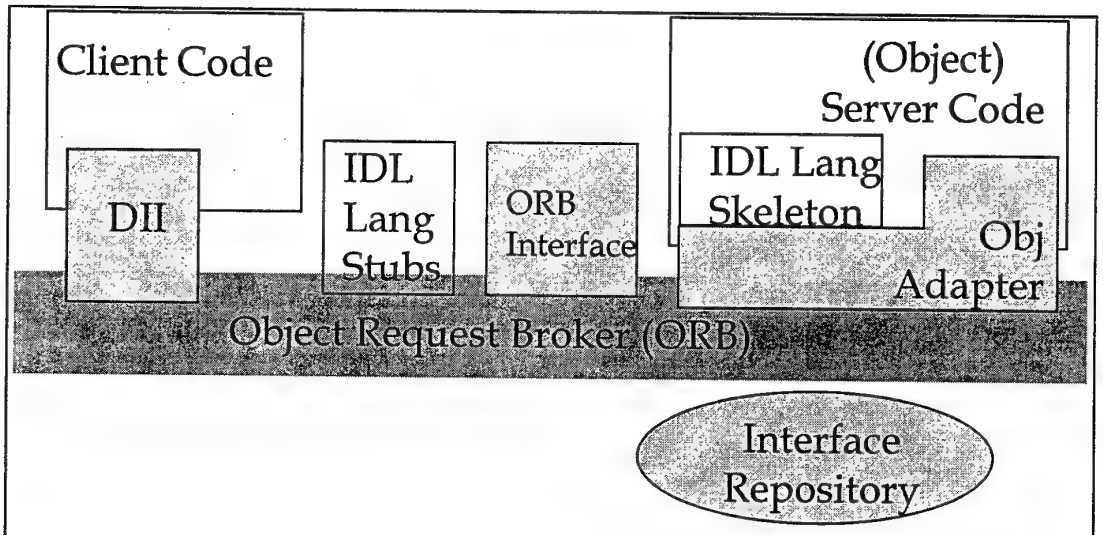

FIGURE 3-11 Interface Repository

(Using the Dynamic Invocation Interface, the client interacts with ORB interfaces (to locate an appropriate implementation), the Interface Repository (to acquire knowledge of the object interfaces), and the server object adapter (to receive a reference for the object to be invoked).)

3.2.3 CORBA Interoperability

3.2.3.1 General Inter-Orb Protocol (GIOP)

General Inter-Orb Protocol provides a generalized, implementation independent specification for composing object requests that are transmitted through the ORB. It consists of a common data representation, standard message formats, and some assumption of the capabilities provided by the transport layer. The protocol element

specifies a standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs. The GIOP is specifically built for ORB to ORB interactions and is designed to work directly over any connection-oriented transport protocol that meets a minimal set of assumptions. It does not require or rely on the use of higher level RPC mechanisms. The protocol is simple ("as simple as possible, but not simpler"), scaleable and relatively easy to implement. It is designed to allow portable implementations with small memory footprints and reasonable performance, with minimal dependencies on supporting software other than the underlying transport layer. While versions of the GIOP running on different transports would not be directly interoperable, their commonality would allow easy and efficient bridging between such networking domains.

3.2.3.2 Internet Inter-Orb Protocol (IIOP)

The Internet Inter-ORB Protocol (IIOP) element specifies how GIOP messages are exchanged using TCP/IP connections. The IIOP specifies a standardized interoperability protocol for the Internet, providing "out of the box" interoperation with other compatible ORBs based on the most popular product and vendor neutral transport layer (TCP/IP). In that sense it represents the basic inter-ORB protocol for TCP/IP environments. The IIOP's relationship to the GIOP is similar to that of a specific language mapping to OMG IDL; the GIOP may be mapped onto a number of different transports, and specifies the protocol elements that are common to all such mappings. The GIOP by itself, however, does not provide complete interoperability, just as IDL cannot be used to built complete programs. The IIOP, and other similar mappings to different transports, are concrete realizations of the abstract GIOP definitions (Figure 3-12).

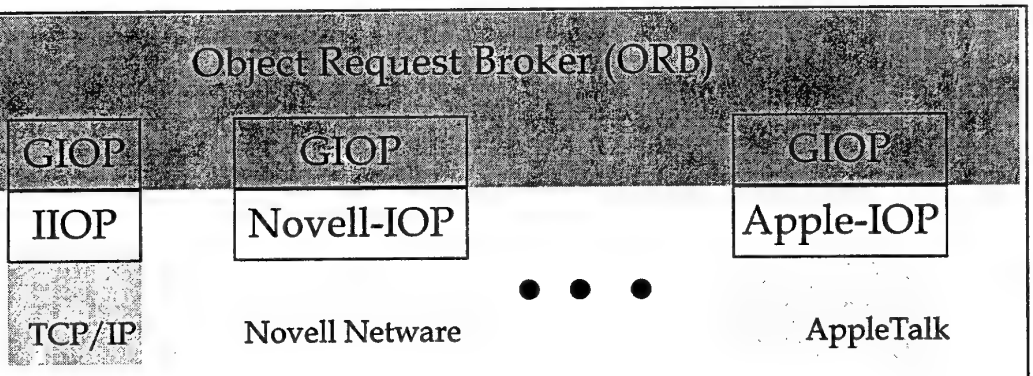

FIGURE 3-12 General Inter-Orb Protocol

(The GIOP sits on top of network specific protocols and provides the common thread for inter-orb communication.)

3.2.3.3 Environment Specific Inter-Orb Protocol (ESIOP)

In order to facilitate incorporation of sites which have substantial existing or legacy distributed environments which use an alternative protocol to TCP/IP because of a design necessity, an Environment Specific Inter-Orb Protocol can be used to connect ORBs of like nature. ESIOP essentially replaces the use of GIOP and a transport protocol (like IIOP). Unlike the GIOP/IIOP pairing, implementation vendors are not required to implement ESIOP. GIOP/IIOP must still be used when the ORB using ESIOP communicates with other ORBs that do not have the same common ESIOP. Currently, the OSF/DCE remote procedure call interface is the only accepted ESIOP by OMG. This OSF/DCE compliant ESIOP is called the DCE Common Inter-Orb Protocol (DCE-CIOP).

3.2.3.4 Protocol Gateways

When vendors implement ORBs with transport mechanisms different than IIOP, ORB protocol gateways must be used to communicate. There are two types of gateways: half-bridges and full-bridges. Half-bridge protocol gateways convert ESIOPs (or other internal transport mechanisms) to GIOP/IIOP. Half-bridges can be implemented without interaction with any other implementation vendor. Full-bridges perform a direct conversion between ESIOPs (or other transport mechanisms). By its nature, full-bridges require the cooperation of both transport implementations (Figure 3-13).

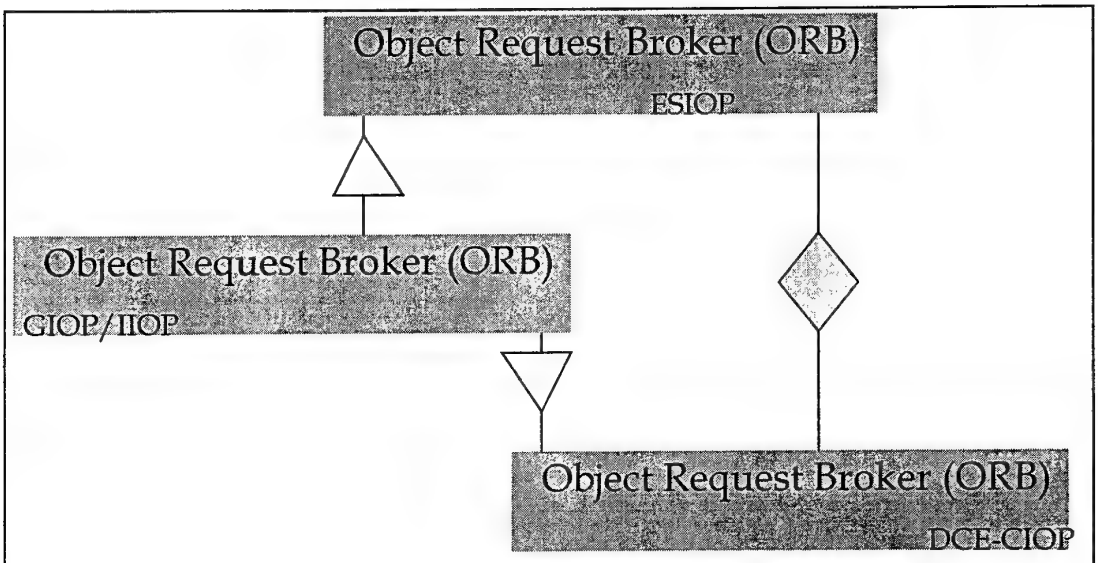

Figure 3-13 Protocol Gateways

(Protocol gateways allow differing ORB implementations to communicate either through half or full bridge architectures.)

3.2.4 CORBA/Interworking

The purpose of the Interworking architecture is to specify support for two-way communication between CORBA objects and COM objects. The goal is that objects from one object model should be able to be viewed as if they existed in the other object model. For example, a client working in a CORBA model should be able to view a COM object as if it were a CORBA object. Likewise, a client working in a COM object model should be able to view a CORBA object as if it were a COM object.

3.2.4.1 CORBA-COM Object Mapping

There are many similarities between the two systems. In particular, both are centered around the idea that an object is a discrete unit of functionality which presents its behavior through a set of fully-described interfaces. Each system hides the details of implementation from its clients. To a large extent COM and CORBA are semantically isomorphic. Much of the COM/CORBA Interworking specification simply involves a mapping of the syntax, structure and facilities of each to the other.

There are, however, differences in the CORBA and COM object models. COM and CORBA each have a different way of describing what an object is, how it is typically used, and how the components of the object model are organized. Even among largely isomorphic elements, these differences raise a number of issues as to how to provide the most transparent mapping.

Still when viewed at this very high level, Microsoft's COM and OMG's CORBA appear quite similar. Roughly speaking, COM interfaces (including Automation interfaces) are equivalent to CORBA interfaces. In addition, COM interface pointers are very roughly equivalent to CORBA object references. Assuming that lower-level design details (calling conventions, data types, and so forth) are more or less semantically isomorphic, a reasonable level of interworking is probably possible between the two systems through straightforward mappings.

3.3 ActiveX/DCOM

3.3.1 Introduction

The theoretical attractions of component-based computing architectures have been well understood for some years now. The basic principle is that applications should be constructed out of component objects whose internal implementation details are hidden and which communicate with other components according to an agreed protocol, possibly by means of public interfaces described in a formal interface definition language and mediated either by a containing document or a "broker" which matches client requests with available services. Conceptually, there are strong parallels between computer peripherals communicating over a standardized hardware bus such as SCSI, using an agreed communication protocol, and software components interacting by means of what is effectively a software bus structure and protocol.

Different implementations of component-based distributed architectures may well provide different procedures for clients to connect to appropriate server components, but the basic principle remains of well-encapsulated relatively independent software entities working together to meet an application's requirements. In meeting client requests, server components may well become clients of other servers (for example, in retrieving information from a database which is then statistically analyzed and displayed in graphical form). Within this paradigm, the term "distributed" is best thought of in the abstract sense as describing components which communicate only via their public interfaces and which desirably exist in separate memory address spaces, rather than in terms of necessitating a physical distribution of the constituents across machines.

The principal benefits to be expected with component-based architectures lie in the areas of interoperability, scalability, flexibility/adaptability, reusability, user familiarity with commonly employed user interface and office productivity components, and speed/cost of application development. In the last year or so, the long-awaited promise of component-based architectures approached fruition with Microsoft extending its OLE and COM (Component Object Model) architecture into a distributed model (DCOM) with ActiveX components. Microsoft has made its application programming interface (API) open to third party developers by giving the standard and source code to the Open Group.

3.3.2 ActiveX and DCOM

A generalized architecture for the component model is shown in Figure 3-14.

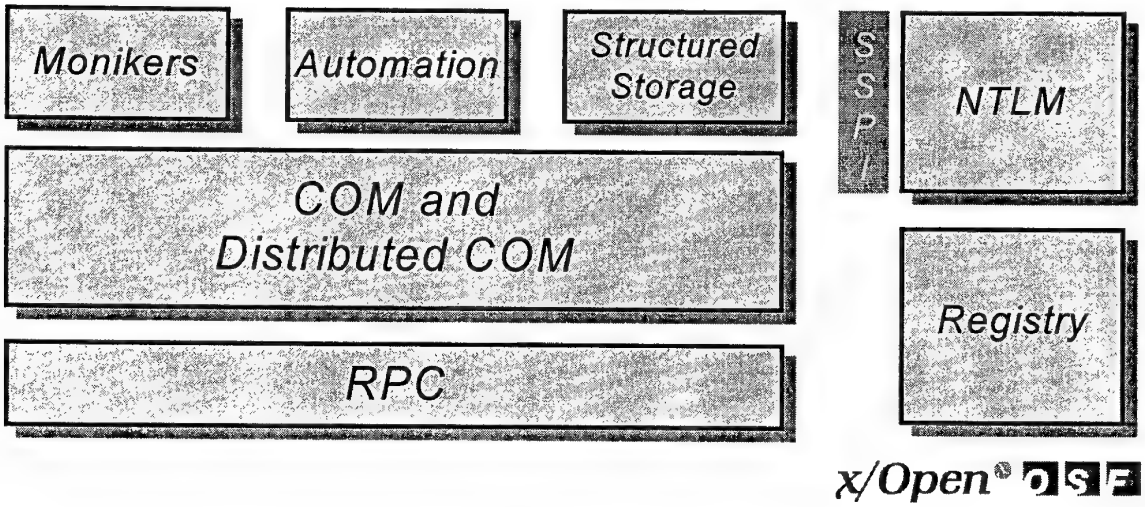

FIGURE 3-14

A generalized architecture for the component object model.

In this diagram the COM and Distributed COM block represents a binary interface specification for components (COM) and the framework for remote activation of COM objects (DCOM). Supporting this block are a number of services:

Structured Storage: Objects need to maintain their state after the program that created them has terminated. A relational database is stateless, i.e. it just preserves data so ActiveX provides a structured storage mechanism to make objects persistent.

Monikers: In order to find services across a network these services must be named. Monikers provide a mechanism to name an object and facilitate binding of that object to other objects.

Security Service Provider Interface (SSPI): An API to generic security services (like the OSF DCE GSSAPI).

Registry: A directory for COM objects.

ORPC: an object-oriented layer on top of DCE RPC that supports remote object invocation.

Since Microsoft gave the ActiveX standard into the hands of the Open Group the Open Group have been working on the convergence of ActiveX/DCOM with OSF DCE. ActiveX DCOM is using an OSF DCE compliant MS RPC and ORPC is built on top of this. In the near future Open Group claims that DCE security will be added to MS RPC and DCOM will be ported to the OSF DCE RPC. The result will be two interoperable implementations which provide full DCE security services under ActiveX.

At a presentation to the COE Working Group in late February Dr. Steve Lewontin, Principal Research Engineer at the Open Group Research Institute claimed that by April 1997 the Open Group, with the assistance of Digital Equipment Corporation, Software AG and SNI, will have ported ActiveX to more than 17 UNIX platforms. Subgoals of this program delivery are that OSF DCE security will be available via SSP and DCE security in MS RPC will interoperate with DCE secure RPC.

By November 1997 they expect to have ported the ActiveX code base to DCE RPC and added Windows NT LM as a DCE security service.

In addition to these developments the Open Group is working with OMG towards integration of CORBA and ActiveX, this is addressed in OMG's ORB RFC 5 which is due out 2nd Quarter 1997

Third party developers have responded to these developments by releasing the core functionality of existing applications as software components. The interactions of these components with other components can easily be specified, for example within the context of a Visual Basic form, to construct complete applications. In the developers' edition of Office 97, Microsoft has made available for use in other applications more than 500 components used in constructing the Office 97 suite. Some of these components are simple user interface controls, but others comprise a major part of the capabilities of the associated Office 97 applications. Lotus is also releasing much of the functionality of the Notes suite in ActiveX component form. Powerful components can usually be "sub-classed" so that they preserve user familiarity (by retaining the look and feel of the full-featured application) and at the same time simplify operations by exposing only the functionality necessary for their intended limited purpose. This approach minimizes user training requirements.

A software component industry is emerging, with vendors who can supply a broad range of components — such as GIS mapping controls, Web browsers, document viewers and Internet connectivity modules — together with various development environments and utilities which simplify construction of component-based applications. Microsoft's ActiveX framework merits close examination, partly because of Microsoft's market dominance but also because of the advanced state of ActiveX in terms of the underlying architecture, the range of existing ActiveX "controls" and the support tools available. Of special importance is the fact that an HTML Web page is one of the range of containers which can host ActiveX controls and allow their interactions to be scripted — this

provides further impetus towards the use of Web browsers as universal user interfaces, with the attendant benefits that this entails. Using Microsoft's terminology, component-based applications are ActiveX containers which host a set of ActiveX controls whose interactions are specified in a suitable scripting language. Using an tool known as the ActiveX Control Pad, Web pages with embedded controls can be put together in a matter of hours to form an application which is activated simply by viewing the page with Internet Explorer v3.0 (or Netscape with an ActiveX plug-in).

Demonstration ActiveX GIS components are available from several Web sites. These can easily be downloaded and embedded in Web pages, together with simple controls to specify the maps to be displayed, to alter the scale, etc. Some of these GIS components are very rich in functionality and require close study to learn how to best exploit this functionality for any given application. Many of the ActiveX controls being marketed commercially are "data aware", meaning that the control can be linked to an associated record in a local or remote database and can reflect (or make) changes in that record. As military developers learn the characteristics of a suitable set of components such as a Web browser, mapping control, database connectivity component and graphical overlay, the construction of new applications will become much less daunting than previously. In fact, once the main functional components of military systems are identified and "wrapped" in code which presents them as ActiveX controls quite complex applications can be constructed quite quickly.

The infrastructure supporting the ActiveX framework is reportedly rather complex, but the complexity is not of itself a central issue as it is concealed from the developer. What is of fundamental significance is the simplicity and power of constructing applications by using containers to host assemblies of well-proven existing components, whose interactions are specified in a simple scripting language. Microsoft's Internet Explorer is reported to have been constructed in this manner — the container, Iexplore.exe is only 38 kilobytes in size! The WebBrowser ActiveX control which provides much of the functionality of Internet Explorer (Mshtml.dll, 831 kilobytes in size) is registered with the Windows 95 system and can easily be inserted within Web pages using the ActiveX Control Pad. (It is tempting to attribute part of Microsoft's success in developing a competitive Web browser within a matter of months to their choice of software architecture!)

Recent developments with Microsoft's ActiveX framework plus the security and other advantages of OSF DCE and potential interoperability with OMG CORBA are significant from two main viewpoints. First, they give substance to the long-standing promise of component-based software development and provide a technology for developers to build real-world applications. Second, with Microsoft's dominance in the marketplace and its obvious commitment to the ActiveX framework, this version of a component-based software architecture is likely to succeed.

3.3.3 Potential Consequences of Component Technologies

What are the consequences for the medium to long-term future of applications development, particularly with respect to interoperability of applications, standards-setting initiatives and end-users? The overall effect should be quite positive, given the ease with which applications could be assembled, modified and extended once a set of suitable components is available. At the most basic level, the "distributed" computing side of the ActiveX framework simply involves retrieval of components from remote sites for local execution. Normal distributed computing at this level occurs, for example, through database and Internet connectivity provided by certain ActiveX controls. At the more complex DCOM level, various distributed ActiveX server components satisfy local client requests. These server components can either run on a local or a remote machine.

In the conventional client/server model of computing, the client and server are separately identifiable machines. In the distributed component model, the physical location of particular components is of less importance — client and server components may even be co-hosted. However, there are clear benefits in terms of interoperability and scalability in writing applications such that client and server components are not particularly dependent on their physical location at execution time.

In the context of COEs it is interesting to look at past efforts at standardizing various aspects of computing and communications technology. These efforts have had a quite profound positive impact on the field of computing, yet some recent standardization efforts have been rendered irrelevant by commercial developments. With the increased pace of change in the computing industry, the objectives and methodology of standardization initiatives need to be re-examined. For example, it may be better to promote standardization by funding reference implementations of protocols, frameworks, application development tools, etc., which are then placed in the public domain, than by requiring compliance with specified standards for software and equipment purchases.

The most notable impact of component-based software should be in the area of user familiarity with applications and consequently with training requirements. Hopefully, the consistency of "look and feel" which exists across the Microsoft Office suite should spread across applications which use the same set of low-level components to implement various types of control boxes, drop-down menus, directory browsers, etc. Similarly, in applications which have Web browsing, word processing or spreadsheet manipulation requirements, "componentised" versions of COTS software could be included in the application's container, with the consequent advantage that users will probably already be familiar with the corresponding commercial packages.

3.3.4 Future Developments

Traditional methods of standards setting have limited value in domains which are undergoing rapid change — developed standards are likely to be overtaken and rendered

irrelevant. Major standardization efforts could well be reserved for domains where there are reasonable prospects of any developed standards being recognized and adopted on an industry-wide basis.

The status of technologies developed by individual vendors, small vendor groups or industry-wide bodies (whether proprietary, in transition to open standards, or genuinely open) should be monitored, and where appropriate military developers should actively participate in the process of developing or enhancing potentially viable technologies with relevance to military systems. (ActiveX/DCOM is an obvious candidate here.) In this context, military C3I projects might become "factories" for specialized military components which could then be used throughout a COE. Such component-based architectures are important because of their ability to facilitate interoperability, rapid application development and reuse of common user interface elements, and to help construct systems which can be adapted for new requirements or re-implemented using new technologies with minimal effort. In this context the most widely used office productivity applications should be recognized as *de facto* standards, but at the same time their vendors should be encouraged to improve overall compliance with COE guidelines. Alternatively, the core functional components of office productivity suites could easily be assembled with customized menus and tool bars which preserved the overall functionality and look and feel of the standard application, yet at the same time improved COE compliance.

4. Description of Representative COEs

4.1 Defense Information Infrastructure-Common Operating Environment (DII-COE)[3]

4.1.1 DII COE Architecture

The DII COE concept is best described as an architecture and an approach for building interoperable systems, a collection of reusable software components, a software infrastructure for supporting mission area applications, and a set of guidelines and standards. The guidelines and standards specify how to reuse existing software, and how to properly build new software so that integration is seamless and, to a large extent, automated.

The DII COE is a "plug and play" open architecture designed around a client/server model. The DII COE is not a system; it is a foundation for building an open system. Functionality is easily added or removed from the target system in small manageable units, called *segments*. Structuring the software into segments is a powerful concept that allows considerable flexibility in configuring the system to meet specific mission needs or to minimize hardware requirements for an operational site. Site personnel perform field updates by replacing affected segments through use of a simple, consistent, graphically oriented user interface.

The DII COE does emphasize both software reuse and interoperability, but its principles are more far reaching and innovative. The DII COE concept encompasses:

- an architecture and approach for building interoperable systems,
- an infrastructure for supporting mission area applications,
- a rigorous definition of the runtime execution environment,
- a rigorous set of requirements for achieving DII COE compliance,
- an automated toolset for enforcing DII COE principles and measuring DII COE compliance,
- an automated process for software integration,
- a collection of reusable software components,
- an approach and methodology for software reuse,
- a set of APIs for accessing DII COE components, and
- an electronic process for submitting/retrieving software components to/from the DII COE software repository.

[3] "Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification, V2.0, Oct 23,1995, DISA JIEO"

Figure 4-1 shows how the DII COE serves as a foundation for building multiple systems. The shaded box shows two types of reusable software: the operating system and DII COE components.

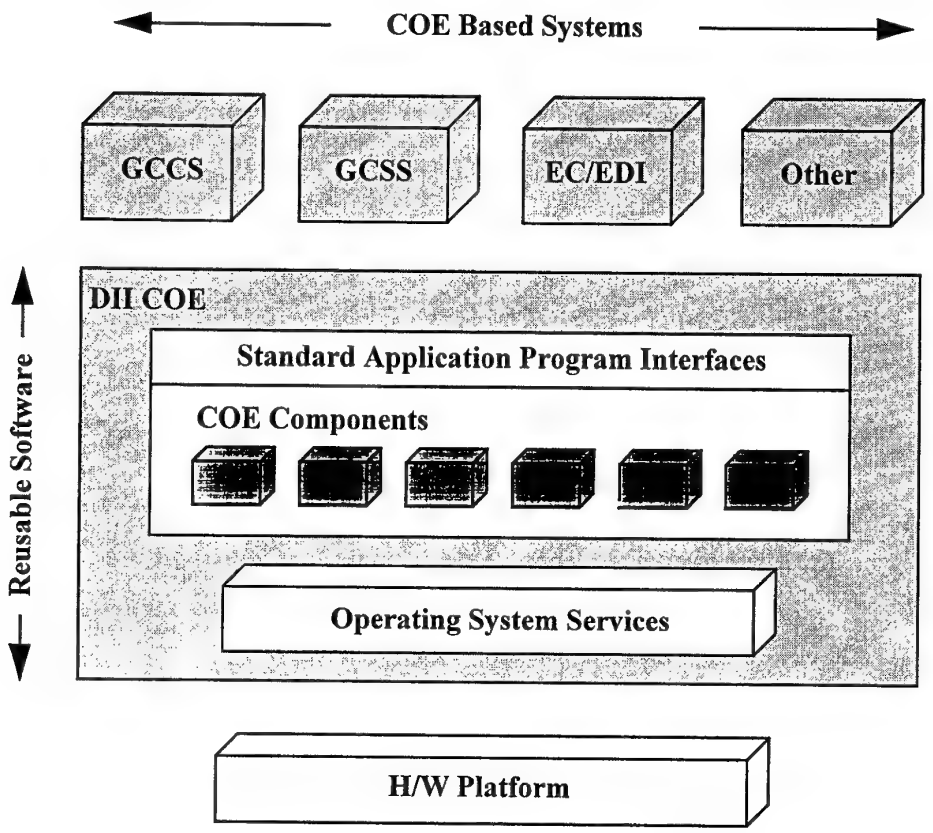

FIGURE 4-1

The DII COE Architecture

In DII COE-based systems, all software - except the operating system and basic windowing software - is packaged in self-contained units called *segments*. This is true for DII COE infrastructure software and for mission application software as well. Segments are the most basic building blocks from which a DII COE-based system can be built. Segments are defined in terms of the functionality they provide, not in terms of "modules," and may in fact consist of one or more CSCIs (Computer Software Configuration Items). The reason for defining segments in this way is that it is a more natural way of expressing and communicating what software features are to be included in, or excluded from, the system than by individual process or file name. For example, it is more natural to think of a system as containing a message processing segment than CSCIs called Icm, Ocm, and Pcm. Those segments which are part of the DII COE are known as *DII COE component segments*, or more precisely, as segments which further

have the attribute of being contained within the DII COE. The principles which govern how segments are loaded, removed, or interact with one another are the same for all segments, but DII COE component segments are treated more strictly because they are the foundation on which the entire system rests.

DII COE databases are divided among segments as are mission applications, but with a different focus. Mission applications are segmented based on their functionality. Databases are segmented by their content or by the subject area of the mission applications they support. Thus mission applications are functional modules; databases are informational modules.

A fundamental principle throughout the DII COE is that segments are not allowed to directly modify any resource "owned" by another segment. This includes files, directories, modifications to the operating system, and modification to windowing environment resources. Instead, the DII COE provides tools through which a segment can request extensions to the base environment. The importance of this principle can not be overemphasized because environmental interactions between software components are a primary reason for difficulties at integration time. By providing software tools which arbitrate requests to extend the environment, integration can be largely automated or at least potential problem areas can be automatically identified.

4.1.2 DII COE Compliance

The degree to which "plug and play" is possible is dependent upon the degree to which segments are DII COE compliant. DII COE compliance can also be addressed by the degree to which integration with DII COE component segments and the runtime environment has been achieved, from "peaceful DII COExistence" to "fully integrated." However, the degree of software integration achieved is only one important way of measuring DII COE compliance. Equally important are measures of GUI consistency, architectural compatibility, and software quality. The DII COE defines four areas of compliance:

Category 1: Runtime Environment. This category measures how well the proposed software fits within the DII COE executing environment, and the degree to which the software reuses DII COE components. It is an assessment of whether or not the software will "run" when loaded on a DII COE platform, and whether or not it will interfere with other segments.

Category 2: Style Guide. This category measures how well the proposed software operates from a "look and feel" perspective. It is an assessment of how consistent the overall system will appear to the end user. It is important that the resulting DII COE-based system appear seamless and consistent to minimize training and maintenance costs.

Category 3: Architectural Compatibility. This category measures how well the proposed software fits within the DII COE architecture (client/server architecture, DCE infrastructure, CDE desktop, etc.). It is an assessment of the software's potential longevity as the DII COE evolves. It does *not* imply that all software must be client/server and RPC based. It simply means that a reasonable design choice has been made given that the DII COE is client/server based and is built on top of a DCE infrastructure.

Category 4: Software Quality. This category measures traditional software metrics (lines of code, McCabe complexity metric, etc.). It is an assessment of program risk and software maturity.

The DII COE defines eight progressively deeper levels of integration for the Runtime Environment Category. Note that levels 1-3 are "interfacing" with the DII COE, not true integration. Integration begins at level 4.

Level 1: Standards Compliance Level. A superficial level in which the proposed capabilities share only a common set of COTS standards. Sharing of data is undisciplined and minimal software reuse exists beyond the COTS. Level 1 may allow simultaneous execution of the two systems.

Level 2: Network Compliance Level. Two capabilities coexist on the same LAN but on different CPUs. Limited data sharing is possible. If common user interface standards are used, applications on the LAN may have a common appearance to the user.

Level 3: Workstation Compliance Level. Environmental conflicts have been resolved so that two applications may reside on the same LAN, share data, and DII COExist on the same workstation as DII COE-based software. The kernel DII COE, or its equivalent, must reside on the workstation. Segmenting may not have been performed, but. some DII COE components may be reused. Applications do not use the DII COE services and are not necessarily interoperable.

Level 4: Bootstrap Compliance Level. All applications are in segment format and share the bootstrap DII COE. Segment formatting allows automatic checking for certain types of application conflicts. Use of DII COE services is not achieved and users may require separate login accounts to switch between applications.

Level 5: Minimal DII COE Compliance Level. All segments share the same kernel DII COE, and functionality is available via the Executive Manager. Boot, background, and local processes are specified through the appropriate segment descriptor files. Segments are registered and available through the on-line library. Applications appear integrated to the user, but there may be duplication of functionality and interoperability is not guaranteed. Segments may be successfully installed and removed through the DII COE installation tools.

Level 6: Intermediate DII COE Compliance Level. Segments utilize existing account groups, and reuse one or more DII COE component segments. Minor documented differences may exist between the *Style Guide* and the segment's GUI implementation.

Level 7: Interoperable Compliance Level. Segments reuse DII COE component segments to ensure interoperability. These include DII COE provided communications interfaces, message parsers, database tables, track data elements, and logistics services. All access is through published APIs with documented use of few, if any, private APIs. Segments do not duplicate any functionality contained in DII COE component segments.

Level 8: Full DII COE Compliance Level. Proposed new functionality is completely integrated into the system (e.g., makes maximum possible use of DII COE services) and is available via the Executive Manager. The segment is fully compliant with the *Style Guide* and uses only published public APIs. The segment does not duplicate any functionality contained elsewhere in the system whether as part of the DII COE or as part of another mission application segment.

The DII COE is continuing to evolve. One major thrust of this evolution are the activities of the DISA/DARPA Joint Program Office(JPO) which is actively sponsoring the infusion of new technology products into the DII COE. Under its Leading Edge Services (LES) thrust it is attempting to extend the common services available under the DII COE by integrating advanced object based client server functionality into the existing DII COE architecture. This will enhance the performance of the DII, extend the functionality, and capitalize on emerging technology while still retaining the same basic "plug and play" concept.

4.2 The UK C2I COE

The C2I COE is an enabling mechanism intended to support a federation of C2I information systems. It is envisaged that systems within the federation will conform to a minimum set of agreements that support the objectives of the COE, but will otherwise be free to evolve to meet the changing requirements of their local user population.
The C2I COE Study Phase 2 Final Report groups the services provided by the COE into the following 11 main areas:

- system services
- network services
- data services
- messaging services
- directory, naming and addressing services
- GIS services
- track services
- OA services
- system management services
- software engineering services
- alert services
- COE integration

The network services area outlines the following requirements for distributed mechanisms within the COE architecture:

- the need to operate over a wide diversity of communications bearer technologies, topologies and grades of service;
- the need to provide effective support for users in a distributed enterprise to work together, including pooling of information and other resources;
- the need to provide effective support for management of the system, including support for reconfiguration in the event of loss or failure of one or more system elements;
- the need for operational flexibility to be capable of meeting the demands of unforeseen deployment scenarios;
- the need to interoperate with other UK (non-C2I) and Allied systems.

Distributed computing mechanisms are perceived to comprise:

- LAN components (including the LAN medium, bridges, etc.); internetwork and transport services that provide the basic network protocol over which all services communicate;
- distributed file services that allow files to be shared between machines in a distributed environment, usually over a local area network. File services allow

individual users to share information between themselves, and also allow corporate information to be shared from centralized locations;
- distributed print services that allow any machine on the network to send documents to any printer connected to any other machine on the network;
- distributed system services that are the analogue of the fundamental system services in a distributed context. These are subdivided into the following:
    - distributed time services (e.g. NTP, DTP) that allow the time on all machines to be synchronized;
    - distributed name services (e.g., X.500, DNS) that map between human friendly names and machine numeric IDs;
    - process distribution services that build on transport services to provide the means to distribute processes across a network and for interaction to occur between those processes;
    - distributed object services that allow objects to be managed in a distributed environment and comprise an object layer interface and an object request broker. The object layer interface defines the characteristics of a distributable object so that applications unaware of its structure can make use of it. The object request broker manages the distribution of objects within a distributed system and mediates accesses between distributable objects and calling applications.
    - distributed security services that allow system security mechanisms provided in standalone systems to be extended across a network by managing security attributes of networked components in a coherent manner. In addition they provide additional protection against vulnerabilities specific to distributed systems by such mechanisms as network authentication.

The notion of an information domain and the mechanisms used to transfer information between domains is central to the developing security framework of the UK C2I COE. Domains might span several systems, and the boundaries between domains might occur within a single system. The main difficulty in defining an acceptable security architecture occurs because of the trend towards large scale interconnection of systems. During the design of the security architecture these problems were faced first, since without a solution here, work on other problems might have been nugatory.

The suggested approach is based around a federation of business oriented domains. Each domain is responsible for its own internal security, though it has some obligations to the federation. By keeping domains relatively small, and providing strong separation between them, it is possible to use relatively lightweight security mechanisms within a domain. These mechanisms are largely provided to help users handle protectively marked information, and can mostly be supported by mainstream COTS products.

If a domain is connected to other domains with lower security clearances, its minimum security requirement is for discretionary labeling within the domain, and for labeled messages for external interaction. The labels are set at the discretion of the user and are used to prevent an honest user from mistakenly sending a message to a domain (or user) which cannot afford it the protection dictated by its marking.

In connecting domains together, the fundamental security principle is that information can only leave a domain on the authority of a member of that domain. This means information can only leave a domain if it is explicitly released, i.e., it can only be pushed out of a domain.

The functionality provided for information exchange between domains could be implemented by simple messaging, sent either explicitly or automatically, or by a shared database. A domain can also offer World Wide Web (WWW) services in a similar fashion to sharing a more conventional database. The WWW server would sit outside of the domain, with information being made available to others when pushed out to the server. Once in the server, the information can be pulled by any client in other domains.

Domains may be aligned along physical network boundaries, in which case connections between domains equate to inter-network connections. Here the threat of attack from external sources can be countered by placing firewall functionality between domains. The firewalls would typically allow messages and database update traffic through, but exclude the use of FTP and TELNET. It may also be necessary to place guards on the interconnections that perform more detailed checks on the outgoing traffic, e.g., checks on the message labels or checks on the message content to ensure the absence of particularly sensitive phrases. One-way filters can also achieve very high levels of assurance, but their use is more limited compared to two-way firewalls. In addition, it is necessary to carefully control the use of one-way filters, because two one-way filters placed in parallel in opposing direction can give rise to two-way communication.

While a domain may be mostly tied to one network, deployable elements may be sited in remote locations and yet still be within the same logical domain. Access to the home system from a deployed element may be by messaging or TELNET. In such cases a form of encryption will be necessary in order to guarantee that the domain's boundaries are not broken, despite the use of physical distribution.

The driving characteristic for system services is the operating system; the embryonic COE adopts two product families, namely UNIX and Microsoft Windows (including Windows 3.1/95, NT). The C2I COE does not rely on the POSIX standards set for portability. Although all major UNIX implementations and Windows NT are POSIX compliant, this generally fails to provide a level of functionality on which useful, portable programs can be written. The COE recommends that effective standards are adopted to promote portability among the UNIX and Windows families (i.e. UNIX93 or 95 and Win32 respectively).

Distributed object mechanisms currently offer a multitude of products and standards of differing maturity and this is one area where the COE must monitor commercial development closely. The likely fruition of object architectures in the commercial world means the recommended medium term target for distributed computing within the C2I COE to be CORBA. In the short term, DCE RPC and ONC RPC are the recommended interim RPC standards. Distributed file services are to be provided by a combination of Sun NFS and the SMB protocols. Interaction between applications on the desktop should use OLE2. For time services on UNIX systems, the Network Time Protocol (NTP) should be used.

4.3 The DND/CF Common User Core Strategy (CUC)

Within the DND/CF an information system architecture modelling framework (Figure 4-2) provides an integrated architectural planning approach consisting of five inter-related architectural views. These views are driven by the organization's business needs (Business Architecture) and employ the enabling capabilities of information technology (Technical Architecture). The framework also provides a representation that allows IM developers of information system to agree on definitions, build common understandings and identify issues for resolution.

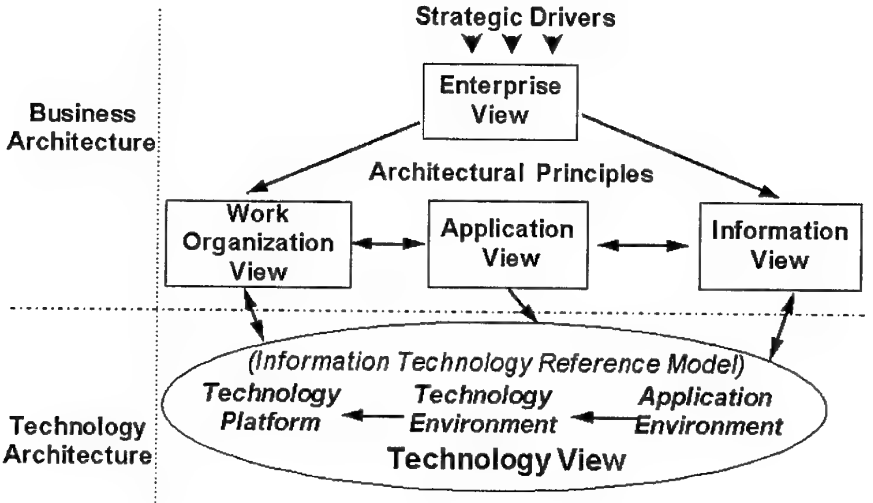

FIGURE 4-2

The DND/CF Information System Architecture Modelling Framework

4.3.1 Information System Architecture Modelling Framework

The Canadian Forces/Department of National Defence (DND/CF) has just completed a major business process re-engineering effort that has resulted in a reasonably coherent Business Architecture that cuts across all functional boundaries and has been the

basis for a 50% cut in all headquarters organizations. This has facilitated the newly formed Defence Information Services Organization (DISO), commanded by the DND/CF Chief Informatics Officer (CIO), in the development of a technology architecture that addresses the needs business architecture in the most efficient manner possible.

The Vision of DISO was to achieve an Integrated Information Environment (IIE) whose primary aim is to enable DND/CF decision-makers with the necessary information they need regardless of their functional position or geographical location in a timely manner. This data centric approach (enabled by the BPR activity), treated information as a corporate resource regardless of who generated it, and the critical success factors were measured through the definition of four Degrees of Interoperability based on the ability to share information rather than connectivity (NATO) or standards compliance (US). [Details concerning the IIE, Degrees of Interoperability etc. can be found in the DND/CF 1995 Information Management Plan] These information sharing requirements necessitated a more extensive solution than the more traditional Common Operating Environments whose goals were often dedicated to reducing the diversity and enhancing the connectivity of the information technology footprint in often selected domains.

The selected approach was called the Common User Core and its extent can be described using the DND/CF Interim Information Technology Reference Model (ITRM). The ITRM (Figure 4-3) blends the NATO Open Systems Environment and US DoD Technical Reference Model with the Canadian Government Information Technology Services model, which is mandated for use by the Canadian Government CIO for all government departments. The DND/CF Common User Core (CUC) essentially provides the majority of the Application Support Environment, the Generic Technology Environment, and the External Environment to include communications and the all important Information (presently classified in Figure 2 as Storage). The CUC supports all DND/CF Business Processes at all Levels of Command (ie. Strategic, Operational and Tactical), and, conceptually, it is analogous to the US DoD Advanced Battlespace Information System (ABIS) Information Grid.

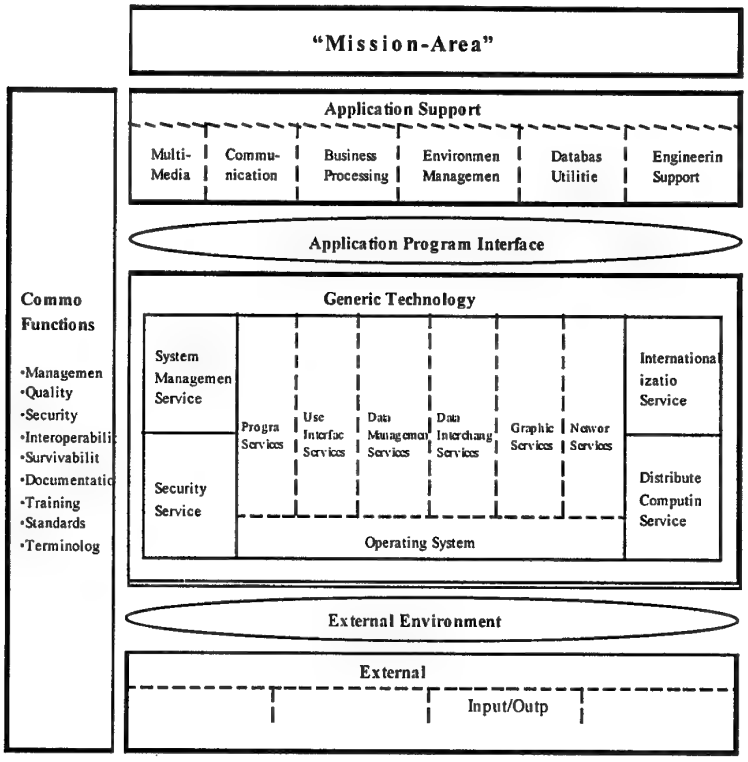

FIGURE 4-3

The DND/CF Interim Information Technology Reference Model

Through the provision of a CUC of essential services, the Information System (IS) Project Management Offices (PMOs) can concentrate on user needs and provide effective decision support Mission Area Applications without having to worry about the underlying technology. As a footnote, the previous absence of this CUC forced the PMOs to spend inordinate amounts of time and effort in the provision of common and redundant services to the detriment and dissatisfaction of the users. The ITRM will shortly be modified to better reflect the external Information Environment needs identified in the Canadian Treasury Board Information Management Services Model and the CCEB Publication 1001 Combined Interoperability Environment: Military Information Management for the Future (issued October 1996).

The Common User Core consists of an integrated suite of standards, products and services. The basic services/projects forming the nucleus of the CUC were determined by the IT team from the BPR activity and were integrated into a DND/CF Long Term

Capital Plan for Information Management (LTCP(IM)) and were approved by the Canadian Treasury Board.

4.3.2 Information Technology Reference Model

Many of the foundation projects, such as the Defence Wide Area Network (DWAN) and the Defence Information Network (DIN) are or will be ready by mid-1997. Follow-on projects, such as the Operational Data Stores, Common Data Exchange Facility and the Data Warehouse, are being accelerated, taking into account the emerging Object technologies (eg. CORBA and DCOM). Initial fielding of these Information Environment projects is expected in early 1998, in time to provide initial operational capability for the new generation of Mission Area Application C2IS for the Joint, Sea, Land and Air environments.

A DISO sponsored Operational Test Evaluation and Interoperability Facility (OTEIF) is being established in Ottawa to enable CUC development and this OTEIF will support the activities of four other OTEIFs concentrating on Mission Area Applications for the Joint, Sea, Land and Air environments. These facilities will involve the users and enable the implementation of the concept of evolutionary acquisition by continuously delivering ever increasing user functionality. As well as being networked to the Canadian OTEIFs, the CUC OTEIF will also act as the link with a NATO Interoperability Facility should the NATO C3 Board follow through on the recommendations of the CIS Business Review AHWG report (1996). A sixth OTEIF for Advanced Technologies (somewhat analogous to the DII Leading Edge Services COE) is still being examined.

The CUC will be dynamic and standards passage, based upon the work being carried out in the OTEIFs, will be through the Defence Information Network (Both the Unclassified and Classified domain), which is an Intranet containing the all-inclusive DND/CF Master Development Plan for Information Management (MDP(IM)) with selected passages being posted to the DISO Home Page on the Internet.

5. Typical Applications

5.1 Global Command and Control System (GCCS) [4]

5.1.1 Background

Since Operation Desert Shield/Desert Storm, the warfighters have increasingly emphasized the need to deploy interoperable command, control, and intelligence; and supporting information systems. The overall concept for the joint move towards interoperability is called "C4I for the Warrior"[5]. Under this concept, the Global Command and Control System (GCCS) was chosen as the single, integrated command, control, communications, computer and intelligence (C4I) system capable of supporting all echelons of the US military command structure.

The implementation plan for GCCS divides it into two components: infrastructure and mission applications. The infrastructure provides all support and establishes the environment required for the proper operation of mission applications. The infrastructure includes hardware platforms, communications connectivity, and software embodied in the Common Operating Environment (COE). Software development, also part of infrastructure, includes integration tools, procedures, configuration management, and other components essential to supporting the functionality and interoperability of mission applications.

Mission applications are software modules designed to perform a specific function or functions critical to mission accomplishment. An example of a mission application is the Theater Battle Management Core Systems (TBMCS), used to develop and disseminate an Air Tasking Order (ATO). Mission applications are developed by separate program offices but need to comply with directives from the GCCS System Program Office to ensure future interoperability and integration into the GCCS Environment.

In addition to improving operations, other goals of "C4I for the Warrior" are to reduce the cost of acquisition, operations, and support. These benefits will be derived through common systems development, system engineering, training, software and hardware reuse and resource sharing. The services intend to take emerging or advancing technology, commercial trends in open system architecture and, Commercial-Off-The-Shelf (COTS) products. When the goals are reached the effectiveness of combat forces to conduct joint operations should be increased significantly and the cost to do it should be reduced.

[4] "Air Force Global Command and Control System Migration Plan, ESC/AV, 1 Mar 1996"
[5] "Command and Control Migration Systems", Letter of 26 Jun 1995

5.1.2 Current GCCS Version 2.1 Technical Summary

GCCS is based on the following components:

- Network connectivity (intrabase and interbase infrastructure)
- Open systems computer architecture
- Common operating environment (COE) with application programming interfaces (API)
- C4I mission applications

5.1.2.1 Network Connectivity

GCCS Wide Area Network (WAN) connectivity is provided by a combination of the DISA Secret IP Router Network (SIPRNET) and the AF Command and Control Network (AFC2N). LAN or other lower level communications are provided at an individual GCCS site by the command responsible for communications support for that site.

A GCCS premise router is part of the GCCS site's LAN infrastructure and normally represents the gateway point out to the SIPRNET WAN. AFC2N interfaces with SIPRNET nodes at many locations.

Additional GCCS communications equipment is needed for sites that host a GCCS Automated Message Handling System (AMHS). This is the Jet Propulsion Laboratory (JPL) AMHS acquired by the Army. Use of this system will require one of several AUTODIN direct interfaces (e.g., AMME, AFAMPE, LDMX, MDT, etc.), a cryptographic device, and a Secure AUTODIN Terminal connected to the local area network. Sites with a high volume of messages may have a dedicated workstation as an AMHS server. One AMHS certification requirement for a AMHS site is that the GCCS LAN, all remote LANs, all backside LANs, all remote dial-in subscribers, and dedicated circuit subscribers must be protected at the Secret classification level.

5.1.2.2 Open System Computer Architecture

Open system computer hardware currently in GCCS includes commercial non-developmental systems offered by SUN Microsystems, Hewlett Packard Co, and personal computer companies. At some sites, PC compatible workstations and Macintosh II FX workstations running X-Windows Server Software are used for X terminal access to Sun SPARC or HP based application servers. Figure 5.1 presents a representative general GCCS hardware and networking configuration.

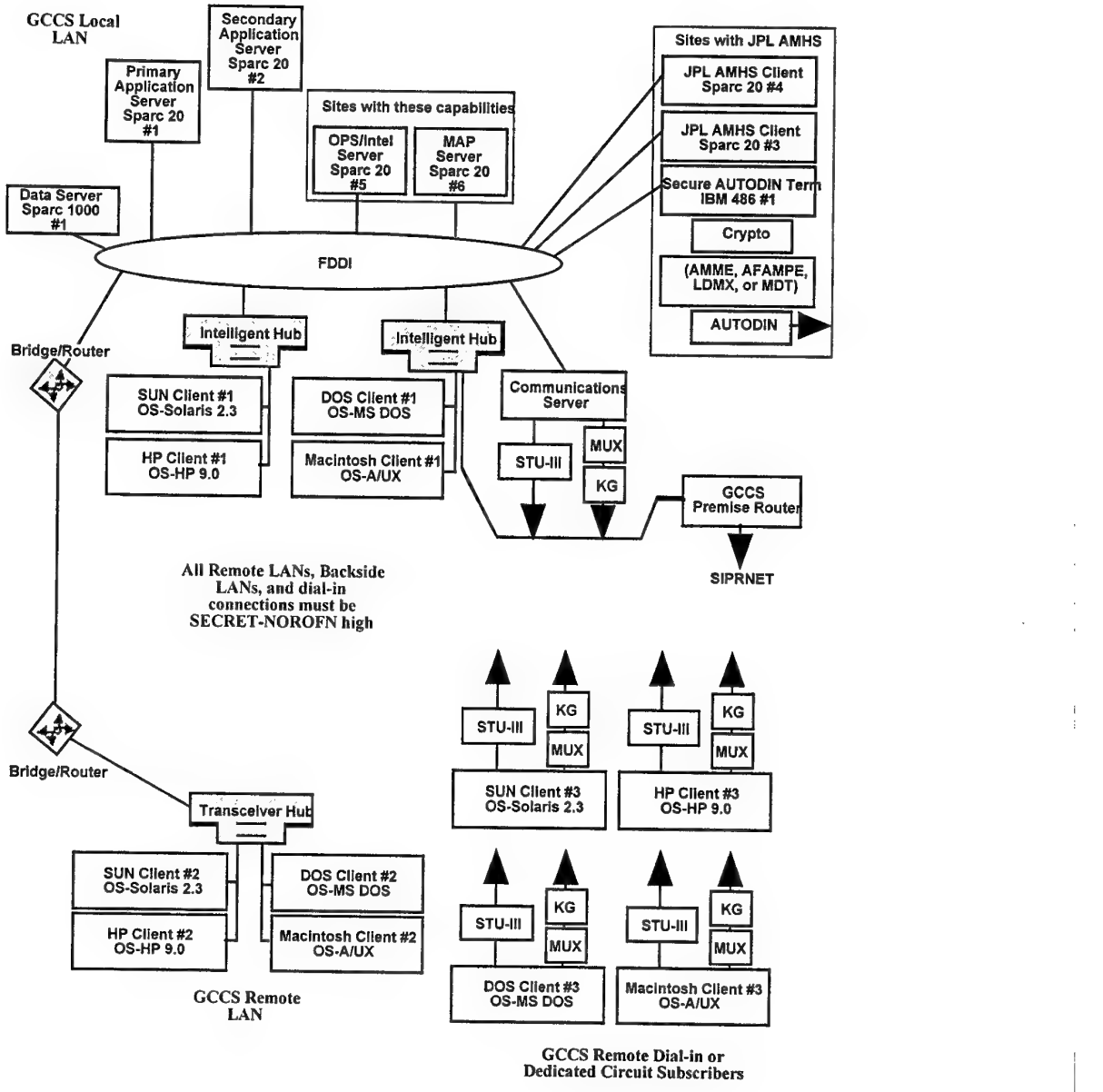

FIGURE 5.1

GCCS General Site Hardware Configuration

5.1.2.3 Baseline GCCS COE and Application Programming Interfaces

The GCCS COE consists of a broad set of mission area applications built on top of the COE. The COE, in turn, consists of a set of common support applications and a set of platform services. Common support applications are centrally provided and offer a set of services that may be widely used in end-user applications, e.g., office automation and on-line help, or common mission applications such as track, correlation, alerts, mapping and message processing. The current version of GCCS, version 2.1, consists of the functional support areas as depicted in Figure 5.2 along with a developer's tool kit.

The COE functions can be accessed through a set of Standard Application Programming Interfaces (API), which in turn support applications. APIs are documented in GCCS programmer's guides which describe the software libraries and services and how to write software modules that interface with and use COE services.

Common Applications: Air Force, Army, Navy, Marines

Standard Application Programming Interfaces

Common Operating Environment				
Network Administration	System Administration	Database Administration	Security Administration	Message Processing
Communications	Correlation	Database Management	File Management	MCG&I
On-line Support	Executive Manager	Alerts	Office Automation	Multi-media Support
Data Interchange	Network Services	Distributed Computing		

Operating System

FIGURE 5.2

GCCS V2.1 COE

5.1.3 Common Mission Area Applications

The major functional areas to be supported by GCCS include the following:

- Deliberate Planning
- Crisis Action Planning

- Mobilization
- Force Deployment
- Force Employment
- Sustainment
- Intelligence

The Defense Information Systems Agency (DISA) is the lead agency for the COE development - technically and administratively. The Joint Staff, through it's "Best of Breed Process for Global Command and Control System", is responsible for selection of mission applications. The four services, the Defense Mapping Agency (DMA), and Defense Intelligence Agency (DIA) participate in the development of both the COE and the mission applications. The Air Force would review and approve the migration of its service-unique applications into the GCCS environment.

5.2 The Departmental Integrated Human Resource System (DIHRS)

The existing human resource information environment within the DND/CF is composed of a number of information systems that deal with personnel information, but have limited abilities to exchange or share information. These information systems had been traditionally separated based on differing requirements (e.g., civilian versus military personnel management, organization versus establishment management) despite similarities within the underlying information. The Associate Deputy Minister (Personnel) (ADM(PER)), as part of its business process re-engineering efforts, is pursuing a initiative to replace five existing human resources information systems with a single integrated information system. This system will integrate active military and civilian human resources processing, and, in the future will also incorporate the reserves.

The DIHRS implementation has been accelerated in order to accommodate the necessary personnel reductions in the ADM(PER) branch which also includes the J1 staff function. As such the immediate selection of a Commercial Off The Shelf (COTS) product that could address most of the business process functionality was required. The selected COTS is based upon the Government Of Canada / Treasury Board PeopleSoft software which is currently being implemented or considered for implementation by 15 other Canadian government departments. PeopleSoft has developed a Canada-unique version of its product and each government department will customize this version to meet their own unique departmental requirements.

The acquisition of this product, along with the potential for many others, has necessitated the development of a flexible information management architecture within DND/CF that can accommodate the integration of COTS products and yet retain the ability to realize the DND/CF Vision of the Integrated Information Environment to transparently share information across all domains at all levels of command. Of principal concern is interoperability with C2IS.

This is a classic case study of where the ideal Information Management architecture/COE had to be altered to accommodate the incorporation of non-standard products in order to meet pressing enterprise concerns, in this case a 50% downsizing in personnel staff. The fact is that the incorporation of Mission Area Applications COTS brings with it an entire supporting cast of Support Applications, Generic Technology and External Environments which has to be closely considered when attempting to maintain the coherence of a COE. A holistic, enterprise business case (with the operational impact as well as affordability being considered) has to be made when acquiring these COTS. In this case the advantages of implementing a wholly integrated human resource system AND permitting a drastic staff reduction [which in turn permitted an increase in combat capability by augmenting the establishment of operational units], outweighed the technical cost of accommodating the solution. The IM lessons are that often non-standard IM solutions are pursued and have to be accommodated by a COE, regardless of technical

elegance, and therefore COE design MUST be flexible enough to permit the reasonable and opportunistic introduction of new IM resources.

6. Non-Technical Critical Factors

6.1 Acquisition Strategy

6.1.1 General.

The mechanisms in place to deliver products to the armed forces of the TTCP member nations are relatively homogeneous in that they are highly regulated and bureaucratized. Although suitable for the delivery of expensive items with a lengthy lifecycle and whose underlying technology evolves gradually, they have led to numerous failures in the adequate and sustained delivery of Information Technology.

6.1.2 Up Front Specifications

Specifically the need to provide detailed Statements of Requirement/Specifications before starting the lengthy Project Approval and subsequent Contracting processes, led to the delivery of obsolete systems as information technology advances outpaced the specifications. More subtly, the initial specifications were based on either manual or semi-automated processes whose automation often leads users to enhance their operational procedures and/or doctrine to take better advantage of the new information technology. This in turn leads to requests to modify the recently introduced system, which inflexible contract specifications, based upon a pre-defined (and often dated) statement of requirement, do not permit, barring prohibitive contract revisions. Not incorporating these innovations leads to user dissatisfaction and a loss of potential combat capability.

6.1.3 New Concept of O&M

The hidden costs of fielding a project are those associated with the bureaucracy of initiation, staffing and justification which often consumes the efforts of personnel with rare IT skills. Furthermore the current acquisition system is based on the delivery of products that can be maintained through a lengthy but finite, pre-defined lifecycle. The nature of information technology is that it is highly dynamic with deliverables that are difficult to predetermine, especially in ill-defined areas such as command and control, and subject to continuous change. Indeed the deliverables continuously evolve to meet new operational requirements. The actual distinction between the delivery of a "system" and its operations and maintenance is almost impossible to discern. In fact there will be an ongoing need to sustain and enhance IT in support of certain capabilities. The notion of projects with a finite lifecycle, should be re-examined to incorporate the notion of projects funding continuous IT services in support of a given capability, such as C2, for as long as that capability is deemed valid.

6.1.4 Use of COTS.

The strength of the information technology industry in the TTCP nations, coupled with an accompanying desire to divest the military of as much as possible of their IT maintenance overhead has led to acquisition practices favoring the use of commercial-off-the-shelf products (COTS). Although superficially attractive, the use of COTS has had drawbacks:

a. COTS are made to support civilian, non-war business processes; their oft-needed modification to suit military needs, has in fact led to the development of a government product that requires specialized, contractual maintenance;

b. COTS products are developed by individual vendors whose products may work well in isolation, but when integrated into a complex enterprise system, may require considerable modification and/or systems integration products, that in turn also have to be maintained;

c. COTS are designed to work in a benign static environment where strict licensing mechanisms are utilized, whereas in a highly dynamic and complex military environment, these mechanisms (such as licensing to specific CPUs or software expiration dates) are unacceptable. In a typically complex (and interdependent) command and control system, the failure of one critical component can cripple the system; and

d. the use of a non-standard COTS product might necessitate COE expansion, along with associated management and maintenance costs, to accommodate the product's particular need.

In summary the use of COTS has many hidden costs that have to be considered before acquiring them. With the proliferation of IT, O&M costs will escalate and the issue of long-term affordability has to be taken into account when rationalizing COTS acquisition.

6.1.5 Contractor Expectations.

Contractor culture will have to be modified to adapt to any revision of acquisition processes. Currently long-term contracts with pre-defined deliverables and costly contract revisions clauses ensure stable cash-flows. When deliverables are not that well-defined, contractors have responded with timely but ill-documented deliverables that for subsequent modifications or maintenance create a dependency of the military on specific contractors. This type of dependency could lead to the delivery of substandard deliverables should the contractors need to be replaced and/or a subsequent stage of work should be competed. The new process will require a mutually beneficial consensus between contractors and the military.

6.1.6 Evolutionary Acquisition.

Evolutionary acquisition is a mechanism used in several nations and alliances (e.g. Canada and NATO) that resolves some of the IT acquisition issues, in particular the issues of staging projects with specifications just preceding contracting for each stage. Although the process is a step in the right direction, there still exists considerable, if not more bureaucratic overhead and authority to use this process is still very much by exception rather than by rule in IT projects.

6.1.7 A New Acquisition Approach.

The dynamic nature of user requirements, the rapid evolution of the capabilities that information technology can deliver, the high cost of information technology and the poor delivery record of IT projects indicate that a new approach to acquisition is sorely needed. Ideally a new, or modified evolutionary acquisition process would have several characteristics:

a. Allow early and continuous user feedback to ensure that the deliverables meet the operational requirement as effectively as possible;
b. Allow for staged implementation with statements of requirements being written before each stage rather than at the beginning of the overall project;
c. Allow for iterative development where subsequent stages will both bring in new functionality as well as enhance or replace existing functionality based upon user feedback;
d. Allow for more general specifications so that IT innovations can be opportunistically incorporated;
e. Allow flexibility to make a business case before acquiring COTS or carrying out custom development, COTS if necessary but not necessarily COTS;
f. Provide for a new understanding between contractors and government; and
g. Builds on the existing foundation of evolutionary acquisition processes.

6.2 Operational Policy/Doctrine

There are several operational factors associated with the migration to a Common Operating Environment (COE). Perhaps they are best summarized by the old adage: Be careful what you wish for - you might get it. Three major issues are addressed below: acceptance of truly decentralized control, security, and the formal definition of operational policy.

A significant requirement of a military COE is that it provide a distributed computing environment. This means that truly decentralized operations are possible. For example, command posts would no longer have to be single geographic entities of people working face-to-face. They could now be expanded geographically and literally spread across continents. The survivability ramifications of this are obvious, but it will require adjustments in attitudes and expectations of control similar to those encountered by the business community as the practice of telecommuting becomes more prevalent. In many ways COEs will make control easier as previously inaccessible interfaces become possible. However, in other ways it may make control tougher as staff members that were previously at an arms length away may now move around based on the situation. This new autonomy will require leaders and colleagues to adjust by making maximum use of limited face-to-face contact to convey intent and to instill trust and confidence in peers and subordinates.

A second factor is security related. If one has a true COE, then anyone else with that same COE can physically connect and interface to it. Although access control mechanisms will be incorporated into a COE, someone has to decide and define what the actual controls will be. This reoccurring process can be complex and tedious, especially when some degree of automation is desired. Although many cases are predefined (e.g., habitual relationships and procedures for operations with alliances), many will still be ad hoc. An especially interesting situation will be interactions with coalition forces where access may have to be decided and defined quickly. Further, defaults policy will have to be re-evaluated. If the default is to not release information (e.g., no-foreign release policy), then it may take considerable time to mark information as releasable and the advantages of having a COE will be lost.

Finally, computers are machines. If true automation is desired, then information will ultimately have to be in a form conducive to machine manipulation. As one moves from a message-based to a model-based paradigm for information exchange, formal models of military operations will have to be created. If information access and exchange is to be automated based upon operational policy, then this information will eventually have to be defined formally. Although the mechanisms can be designed by computer scientists and engineers, the actual policies must be defined by military and political experts. This is an intellectually taxing process that requires a significant contribution by senior domain experts.

6.3 Enterprise Considerations

The ideal implementation of a COE will be realistically constrained by Enterprise-wide considerations. With the militaries of the TTCP nations undergoing downsizing and re-engineering as well as committing forces in numerous hazardous operations other than war worldwide, the implementation of a pervasive COE will be neither simple nor rapid. The following key factors will have to be considered in the establishment of a COE:

a. Operational Requirements. Rapidly evolving situations or enterprise determined immediate operational requirements might require the fielding of non-COE compliant solutions. This is especially true when the COE is in the early stages of deployment and ill-defined; leading to the adoption of de facto standards. It is also inevitable that when a COE satisfies the vast majority of user needs, it is unaffordable and impractical to extend the COE to support highly specialized systems; so there has to be a case for COE exceptions. What is important, and non-negotiable, is that the information these systems generate be entered into the COE Information Environment so that all authorized enterprise users have access to them;

b. Legacy Systems. All of the militaries have critical legacy systems that are not compliant with current COE architectural practices, but are crucial to the operation of the enterprise. Often they are quite adequate, maintenance intensive but not worth replacing in view of overall IT priorities. The engineering challenge is to ensure, as mentioned above, that the information they generate is captured and shared.

c Budgetary Reality. Although the COE will in the long run save costs, the initial costs, including the hidden planning, architectural and managerial (including configuration management) costs, will be significant. The implementation phase will require both considerable fiscal and trained human resources that the enterprise will have to invest before seeing any savings. These facts also often dictate a phased introduction of any COE; and

d. A COE for Combined Operations. All militaries have designed their COEs to address their operational and domestic needs, including industrial benefits. It is wishful to think that all nations will adopt one COE in order to ensure interoperability for Combined operations. Rather the critical feature for external interoperability is to enable the transparent and timely sharing of information using a minimum set of standards that form the interoperability basis of national COEs. The NATO Army Tactical Command and Control Information System (ATCCIS) is an example where an information sharing architecture has been established using a minimum set of standards, that still permit nations to have their own unique COE. To a certain extent the challenges inherent in establishing a COE for Combined Operations is not unlike one that is required to accommodate non-standard COTS products.

6.4 COE Maintenance/Management

A COE can be defined as the specified set of infrastructural software standards and components used in support of an application development, operational deployment, and information interchange with other applications. A COE usually attempts to limit the choices of the diverse set of infrastructural components available in order to provide a standard consistent and cost effective set of information services for the applications that use it. This definition would seem to imply that a COE is statically defined and that it does not change. This is not the case. With very few exceptions the changes in technology found in the COE's COTS, GOTS, and Standards force an evolution of the COE through versioning and the evolution of the dependent applications through levels of compliance. Adherence to a COE by an application is described as its' level of compliance. Given this, the management of a COE involves the management of the policies defining how a COE will change, how non-compliant applications can migrate to the COE. There needs to be a continuous maintenance of the coherency of the COE in that it needs to be complete, be functionally supportive, and well documented. Changes to the COE can have a large affect on the applications ability to evolve and function, which will in turn have a direct impact on the acquisition and fielding costs.

Distributed configuration management of deployed applications made up of COTS, GOTS and software patches over coalitions spanning global distances and including mobile nodes requires close coordination and agreement on how and when to distribute implement changes. This large sustained base in turn affects the speed with which the COE can evolve to incorporate new technology. Thorough testing and debugging of applications built on a COE is needed to ensure that new components will work with the existing base in the various fielded configurations it is expected to work in. In a global distributed environment that diversity includes mobile nodes, resource rich sites such as a supercomputer centers for modeling and simulation or a resource constrained site such as a small handheld device. As more distributed interactive applications are developed the issue of what is a "typical" site becomes an issue that is hard to test, debug, and manage. The COE must support the richness of these environments.

In order to manage the lifecycle of components within the COE there may be specified "Sunset Clauses" in order to phase out or extract and transition components out of the COE for various reasons. This gives the overall sense that lifecycle costs are in the hands of the COE developers. The often unaddressed, difficult or impossible to plan for aspects of the COE lifecycle costs are the forced upgrades or termination of a COTS product or its support brought on by the vendor. These unilateral moves made by the vendor based on its' market strategy leaves the Coalition or one of its' members with the need to replace, upgrade, or take on additional maintenance costs that were unplanned. Much of the costs associated with this involve the testing, debugging, configuration control and timed deployment of a new or altered product. A simple change in a vendors

licensing policy can have a drastic affect on the development, evaluation or deployment cost for the various applications based on the COE.

Often a COE will chose a particular vendors product not only because it is "state-of-the-art" technologically but because the vendor has made commitments to meet a particular industry or governmental standard which is of value to the Coalition. The "as advertised" compliance to a given standard is not always to the 100% level at any given point in time and may vary with versions over time. The standards compliance issue is called out by the COE as an attempt to remain resilient to changes in corporate products over time allowing the COE to chose a different vendors product if logic or cost dictate it. While a vendors product may meet a given standard or specification the vendor is free to provide additional enhancements, extensions, or features to their product. These additional features are often very advanced or desirable to the applications builders who use the COE. If these non-standard extensions are used by an application it leaves it bound unable to switch from one vendors "standard" product functions to another. The acquisition costs in fixing or changing the applications can leave the COE "standardized" on a particular vendors product.

Maintenance of a data or object model across the COE operational base requires a universal consistence. However, maintenance of consistency often causes a decrease in the overall COEs' ability to respond to the often rapid, dynamic or incremental changes required for the operational deployed base to respond to changes in its environment. If the description and data schema used to describe a new type of target are not consistent with the Coalitions data schema it can prevent the interoperation of applications. The way in which new data types are described and incorporated across the application base must be dynamic and must be supported by the COE. The issue of data interchange inconsistencies and incompatibilities is made even more difficult when you consider the cross agency application domain interactions that are a part of the increasingly global and coalition force operations. The description and internal representation of a "truck" can be very different country to country and agency to agency. The Red Cross database and the World Health Organization database may not match up with a particular countries defense department.

6.5 Evolution

Evolution is the way in which the COE changes over time in the light of multiple factors which impact upon it. The COE's ability to adapt to changes in the arena in which it operates is an important factor in it's success. Some of the factors that will impact on the COE and should be taken into consideration include:

Change of Operational Requirement - An example of this is the changing nature of operations towards peace keeping and peace making, which generate the requirement for temporary coalitions with a wide variety of nations. This future vision of military operations is outlined in [1] in the following way: "Organizational and command structures will be flexible, varying according to operational necessity and national commitment. Typically the composition of any force will be significantly different from any previous deployment, subject to intermediate political pressures and requiring the co-operation of forces that have had little or no experience with each other."

This future vision dictates that the COE must be capable of adapting to meet these operational requirements as they arise. This requires infrastructural flexibility which is defined in [2] as "...the degree of adaptability of the COE and its supporting elements to changing circumstances within the business." This flexibility is a vital factor of the COE in helping to ensure success in the changing arena of future operations.

Change in Organizational Structure - An important factor in the deployment of a COE is the effect it has on the traditional military hierarchical structure. As the COE evolves and increases in functionality, the users of the system have greater access to information traditionally denied to them. This is stated in [1] as "Future information services are bypassing the traditional structures, providing information access to all levels and empowering individuals to participate in the planning process and conduct of operations irrespective of their organizational position". This change must be effectively managed in order to maximize the potential of a collaborative process and to allay the fears of those who have been used to the traditional hierarchical approach.

Technology Advances - Technology advances in information systems are becoming ever more rapid and development time reduced. This has been shown recently with the advent of the World Wide Web (WWW) and subsequently JAVA, which have swiftly been integrated into many COTS products. This has two implications within a COE. Firstly revisions of COTS products will be based on or include new technologies and hence as applications are upgraded within a COE, these advancements will become part of the system. It is possible to continue to use an older version of the software to avoid the issue but this has serious implications as the software vendors discontinue support for the older versions. Secondly, with the widespread take-up of IT in the home, users within the COE environment have become accustomed to taking advantage of the latest technologies and will expect to see the same applications within their work environment.

In the light of these rapid advances, the COE should be specified in such a way as to effectively manage the introduction of these technologies as they appear.

Training - As the COE changes over time, there will be a requirement for a coherent training program to ensure the continuing efficient operation of the system. This is not just the case for technology changes, but also for changes in policy which inevitably impact on the users of the system.

7. Technical Deficiencies

7.1 Low Bandwidth Substrate

COEs are based upon commercially available distributed computing environments that expect generous bandwidths, typically via LANs. Although LANs and WANs are common in military environments, there is a large set of users, typically at the lower echelons, whose communication environment is wireless and dynamic. In these cases, bandwidth is often measured by hundred's rather than millions of bits per second. To cope with this problem, low echelon users have their own command and control systems based upon passing procedural messages; there is no concept of a distributed computing environment. If a COE is developed without the capability to adapt to low bandwidth environments, it will not be adopted by this large set of users and a significant portion of the battle command community will be excluded from the advantages of a COE. Lightweight protocols with the ability to measure channel (or network) performance are required so that the COE can automatically adapt to widely varying communications capabilities. If communication independence is not included as a feature of the COE, then a large fraction of military users will continue to build specialized command and control systems with their own set of interoperability problems.

7.2 Security

The containment of information according to the need to know is a necessary part of any operational system, however a major reason for the existence of a COE is to easily exchange information. Security can easily be an impediment to this exchange. Sometimes this is due to policy and sometimes to the lack of a technical solution to the problem.

A policy such as the US policy that documents should have a NOFORN default classification creates problems in operations with coalition partners that technology cannot solve even when it is vital that information should be exchanged. Such a policy is in essence arbitrary and could be changed but the ramifications of such a change would be widespread and would require a change of mindset. A similar situation occurs between domains within the national services, say Army and Navy, and if information is to be freely available within a COE to whom so ever needs it a re-evaluation of classifications and the need to know would have to take place.

Whilst this problem is not easy to solve a more difficult problem is providing the technology to securely transfer information across the boundaries between security levels. This occurs in information flow in both directions. Probably the best example occurs in intelligence organizations. These organizations are necessarily Top Secret but need to connect directly to unclassified, open source material and also to transmit Secret

intelligence to the operational HQs. The problems differ depending on the direction of information flow but before a COE can function effectively there needs to be a technical solution to Multi Level Security.

There are many other problems to address. These include: secure authorization of users, rapid encryption of information such as imagery over secure communication lines and secure configuration and version management. The old rules and technology need to be carefully scrutinized so that those framing a COE are risk aware rather than risk averse.

7.3 Mobile Nodes

COE users, especially at the Tactical level of command will be constantly maneuvering and either leaving the COE or switching between different COE transport layers. Maintaining information accuracy/currency and address consistency are major difficulties. Another aspect will be the need to examine the scalability of the COE to accommodate the environment of the mobile nodes.

7.4 Information Holding

The current loose federation of databases is inadequate and will hinder the critical sharing of common information essential for operations. An information architecture is required to ensure that data, information, objects and eventually knowledge is deliberately stored in a manner that will make these products available in a timely manner to COE clients. An Information Architecture will ensure that information products are consistent, accurate, replicated and protected. It will facilitate the development/addition of Mission Area Applications which can reuse or interact with the standard interfaces which are a critical component of the information architecture. These interfaces will also make information sharing/interoperability with foreign and non-military organizations easier.

7.5 Interface to Non-COE Components

The best COE may achieve 95% integration of the components needed to support an application domain leaving 5% of the components unaddressed. The remaining components need to be addressed through interface extensions so that products from these areas can be used and exchanged. Examples of this type of deficiency can be found in the area of large supercomputing requirements in support of both the modeling and simulation, and imagery communities. COTS chosen for a COE because they address generic requirements may not be ported to special platforms due to low market share/profitability. At the other end of the spectrum, there will continue to be numerous small devices which do not warrant the full COE approach due to their

specialized/limited function but which need to deliver and receive products from the COE base.

7.6 Survivability

Survivability within a COE is predicated on the ability to make use of resources at an alternate location to rehost a function that ran on a node which is no longer available. The distribution inherent in a COE provide for enhanced survivability, however, it also implies the need for functionality that current COEs do not possess. Replication mechanisms which can instantiate and maintain consistency for copies of critical modules must be available. Adaptive fault tolerance mechanisms which can sense the failure or anomalous behavior within the COE, and dynamically create the bindings to the replicated modules must be incorporated. This dynamic movement of data and processes among the nodes of the COE imply the need for resource monitoring and allocation mechanisms which can perform resource management on a global basis across the COE.

7.7 Time Dependency

To effectively use a COE for military applications it must have the ability to handle time critical processing such as tracking and responding to tactical missiles. This entails a number of specific issues beginning with the extension of the object model to incorporate temporal attributes. The invocation mechanisms must also accommodate the fact that several distributed objects have interdependent time constraints. As complex operations execute across multiple nodes of the COE, capabilities must exist to handle the distributed control threads, and the unique error conditions that can arise with failed time dependent operations.

7.8 Conformance Testing

While required and essential for system development, conformance testing is underutilized in many of the potential COE-candidate products. Conformance testing refers to a series of tests which are applied to give confidence that a software, hardware, or combination system fulfills its intended function against whatever measures are applied (accuracy, performance, usability, interoperability, etc.). As COEs are created, procedures must also be established to test not only the individual components but also the aggregate function. Industry tools and methodologies to assist in defining requirements, designing functions, and evaluating results are either ad hoc (at best) or non existent (at worst).

7.9 Formal Interface Specification

A major feature of a COE is the interface specification. A formal interface specification has two facets: architectural and behavioral. The architectural component describes the structure of the data that is exchanged while the behavioral component describes the process (or procedures) for information exchange. For example, in a communications protocol, the structure of the header is an architectural component while the process for acknowledging receipt of information is a behavioral component. Current distributed computing environment specifications have a formal architectural component but not a formal behavioral component. The architectural components are defined using a formal, unambiguous interface design language while the behavioral components are written in English, an extremely ambiguous language. To provide an unambiguous interface specification, the behavioral component must be as unambiguous as the architectural one. Consequently, the interface design language must include facilities to specify the behavioral as well as the architectural components of a system. However, the IDL must still remain implementation independent.

8. Recommendations & Conclusions

This study has considered the current and emerging DCE/COE technology in the context of their potential application in military systems. It was motivated by the explosive growth of activity in the commercial sector, as well as the potential for significant enhancements that these systems offer.

Among the potential benefits are:

1. interoperability among large collections of heterogeneous computing elements based on standard system interfaces
2. uniform "look and feel" across large distributed information system infrastructures based on common application programming interfaces
3. maximize the use of commercial-off-the-shelf products which embody the technology
4. enhanced capability to incorporate new technology to accommodate changing requirements
5. ability to scale systems to accommodate changes in user community
6. increased survivability through reconfiguration mechanisms which ensure that critical functions are allocated to surviving resources
7. provide the necessary architecture/infrastructure to support a uniform shared data and object environment.

COE technology is rapidly maturing in the commercial community. New products are constantly reaching the market with differing levels of functionality, maturity and robustness. Examination of the current development status leads to several conclusions:

Conclusion 1: Current offerings are suitable for experimentation and evaluation in collaborative testbeds, as well as for use in small carefully "crafted" applications. They require additional functionality (e.g. replication, support for time dependent processing, etc.) as well as added stability and robustness before widespread application in large scale systems will be practical. Continued development needs to occur to bring this technology to fruition.

Conclusion 2: Since there is not, and probably will not be a single dominant COE architecture, interoperability between COEs is, and will continue to be an issue. Military users in the member nations should assume that the future will be a heterogeneous environment of many vendor offerings built against several different architectural models. The models may have significant similarities in key areas (e.g. object orientation, client server model, etc.) but will have different implementation strategies, different functional characteristics and different application programming interfaces.

The key to effective use will be standard interoperability/ interface definitions between COEs.

Conclusion 3: The COEs used in the national systems will be specific to that nations applications representing a critical example of heterogeneous COEs. To provide interoperability between the COEs will require the solution of several significant issues both technical and nontechnical. The major COE developers have recognized this need and are currently working on the solutions to the technical interoperability. It remains to the member nations to address the non-technical issues, for example, releasibility and data exchange issues.

It must be recognized that the military are not the primary market drivers in the development of COE technology. However, the military have requirements which may not be met solely by the market drivers of the commercial sector. To address this issue, two recommendations are made:

Recommendation 1:

The military should concentrate its development resources in this area on specific COE functionality needed to support its operations. These include:

a. current COE development assumes high performance networks as the communication substrate on which the COE is built. To support the integration of lower echelon units into the command and control system built on the COE architecture, it is necessary to extend the COE to be able to function over low bandwidth communications such as the combat radios prevalent in the operational community

b. extend COE capability to be able to support mobile nodes as part of the configuration

c. security issues associated not only with access control, authentication and auditing, but also operation within a "firewalled" environment and enforcement of coalition releasability policy

d. to achieve the inherent COE potential for enhanced survivability it is necessary to develop new resource allocation mechanisms. These mechanisms must be able to dynamically assign processes and data to various nodes of the configuration and maintain consistent, replicated copies of critical items. In response to faults or failures the capability must be available to dynamically reassign the critical process or data to alternate nodes which continue to function correctly.

e. to support the dynamic needs of command and control applications (e.g. time critical targets) it is necessary for the COE to support time dependent processing. This implies more than "fast" processing, but rather the fact that the value of the process completion is a function of the time parameters associated with the

activity(e.g. complete before time X, complete after time X, complete after time X but before time Y, etc.). The complexity of this type of processing significantly increases in a large distributed information system.

Recommendation 2:

To insure that the military unique developments are incorporated into the evolving COE standards, it is recommended that the representatives of the military R&D community actively participate in the standards groups, the Object Management Group and the Open Group.

Recommendation 3:

As indicated above there currently are COE offerings which are sufficiently mature to be used for experimentation and small applications. It is recommended that S Group undertake an experimentation/evaluation program involving STP-8, STP-9 and STP-10. This effort would involve STP-10 hosting a COE(s) on the ACCORD testbed, with STP-9 using this configuration as a platform for implementing selected coalition applications. This effort would allow "hands on" investigation of issues of interoperability, security, and performance, as well as assessing the level of functionality of the selected components, the adequacy of the application programming interfaces and the robustness of the selected components.

Recommendation 4:

The panel found that many of the projected plans of the major COE developers have major implications for future military applications. It is therefore recommended that the tracking and assessment of COE evolution be continued by STP-10. If the joint experimentation proposal is accepted, the continuation of the tracking and assessment activity would provide valuable input to the evolution of that testbed. An addendum to the report will be generated during the next year to capture the key advances in the technology as well as results of any collaborative experimentation.

9. Appendices

9.1 CORBA Services and Facilities

9.1.1 OMA Object Services (CORBAservices)

Object Services standardize the life-cycle management of objects. Interfaces are provided to create objects, to control and track objects, to manage access of objects, and to manipulate the relationships between styles of objects (class management). Object Services provide for application consistency and increased programmer productivity. Object Services are broken into two classes: adopted specifications and works in progress.

9.1.1.1 Adopted Object Services

9.1.1.1.1 Naming Service

The Naming Service provides the ability to bind a name to an object relative to a naming context. A naming context is an object that contains a set of name bindings in which each name is unique. Name resolution determines the object associated with the name in a given context. Through the use of a very general model and by dealing with names in their structural form, naming service implementations can be application specific or be based on a variety of naming systems currently available on system platforms. Graphs of naming contexts can be supported in a distributed, federated fashion. The scaleable design allows for a distributed, heterogeneous implementation and the administration of names and name contexts. Because name component attribute values are not assigned or interpreted by the naming service, higher levels of software are not constrained in terms of policies about the use and management of attribute values. Through the use of a "names library", name manipulation is simplified and names can be made representation-independent thus allowing their representation to evolve without requiring client changes.

9.1.1.1.2 Event Service

The Event Service provides basic capabilities for asynchronous events (decoupled event suppliers and consumers), event "fan-in", notification "fan-out", and, through appropriate event channel implementations, reliable event delivery. The Event Service design is scaleable and is suitable for distributed environments. There is no requirement for a centralized server or dependency on any global service. The Event Service interfaces allow implementations that provide different qualities of service to satisfy different application requirements. In addition, the event service does not impose higher level

policies (e.g. specific event types), allowing great flexibility in usage with a given application environment.

Both push and pull event delivery models are supported: that is, consumers can either request events or be notified of events, whichever is needed to satisfy application requirements. There can be multiple consumers and multiple suppliers of events. Suppliers and consumers can generate and receive events without knowing the identities of the others. The event channel interface can be subtyped to support extended capabilities. The event consumer-supplier interfaces are symmetric, allowing the chaining of event channels, for example, to support various event filtering models. Event channels can be chained by third-parties. Typed event channels extend basic event channels to support typed interaction.

9.1.1.1.3 Life Cycle Service

The Life Cycle Service defines conventions for creating, deleting, copying and moving objects. Because CORBA-based environments support distributed objects, life cycle services define conventions that allow clients to perform life cycle operations on objects in different locations. The client's model of creation is defined in terms of factory objects. A factory is an object that creates another object. Factories are *not* special objects. As with any object, factories have a well-defined OMG IDL interface and an implementation in some programming language. The interface for a generic factory, allows for the definition of standard creation services, and, a LifeCycleObject interface defines remove, copy and move operations.

9.1.1.1.4 Persistence Service

The Persistent Object Service (POS) provides a set of common, open interfaces to the mechanisms used for retaining and managing the persistent state of objects. The object, itself, ultimately has the responsibility of managing its state, but can use or delegate to the Persistent Object Service for the actual work. While there can be a variety of different clients and implementations of the Persistent Object Service, all can work together.

9.1.1.1.5 Transaction Service

The Transaction Service supports multiple transaction models, including the flat (mandatory in the specification) and nested (optional) models. The Object Transaction Service supports interoperability between different programming models. Network interoperability is also supported, since users need communication between different systems, including the ability to have one transaction service operate with a cooperating transaction service using different ORBs. The Transaction Service supports both implicit (system-managed transaction) propagation and explicit (application-managed)

propagation. With implicit propagation, transactional behavior is not specified in the operation's signature. With explicit propagation, applications define their own mechanisms for sharing a common transaction. The Transaction Service can be implemented in a TP monitor environment, so it supports the ability to execute multiple transactions concurrently, and to execute clients, servers, and transaction services in separate processes.

9.1.1.1.6 Concurrency Control Service

The Concurrency Control Service enables multiple clients to coordinate their access to shared resources. Coordinating access to a resource means that when multiple, concurrent clients access a single resource, any conflicting actions by the clients are reconciled so that the resource remains in a consistent state. Concurrent use of a resource is regulated with locks. Each lock is associated with a single resource and a single client. Coordination is achieved by preventing multiple clients from simultaneously possessing locks for the same resource if the client's activities might conflict. Hence, a client must obtain an appropriate lock before accessing a shared resource. The Concurrency Control Service defines several lock modes, which correspond to different categories of access. This variety of lock modes provides flexible conflict resolution. The Concurrency Control Service also defines Intention Locks that support locking at multiple levels of granularity.

9.1.1.1.7 Relationship Service

The Relationship Service allows entities and relationships to be explicitly represented. Entities are represented as CORBA objects. The service defines two new kinds of objects: relationships and roles. A role represents a CORBA object in a relationship. The Relationship interface can be extended to add relationship-specific attributes and operations. In addition, relationships of arbitrary degree can be defined. Similarly, the Role interface can be extended to add role-specific attributes and operations. Type and cardinality constraints can be expressed and checked, exceptions are raised when the constraints are violated. Whereas the Life Cycle Service defines operations to copy, move, and remove graphs of related objects, the Relationship Service allows graphs of related objects to be traversed without activating the related objects. Distributed implementations of the Relationship Service can have navigation performance and availability similar to CORBA object references. Role objects can be located with their objects and need not depend on a centralized repository of relationship information.

9.1.1.1.8 Externalization Service

The Externalization Service defines protocols and conventions for externalizing and internalizing objects. Externalizing an object is to record the object state in a stream of

data (in memory, on a disk file, across the network, and so forth) and then be internalized into a new object in the same or a different process. The externalized object can exist for arbitrary amounts of time, be transported by means outside of the ORB, and be internalized in a different, disconnected ORB. For portability, clients can request that externalized data be stored in a file whose format is defined with the Externalization Service Specification. The Externalization Service is related to the Relationship Service and parallels the Life Cycle Service in defining externalization protocols for simple objects, for arbitrarily related objects, and for facilities, directory services, and file services.

9.1.1.1.9 Query Service

The purpose of the Query Service is to allow users and objects to invoke queries on collections of other objects. The queries are declarative statements with predicates and include the ability to specify values of attributes, to invoke arbitrary operations, and to invoke other Object Services. The Query Service allows indexing, maps well to the query mechanisms used in database systems and other systems that store and access large collections of objects, and is based on existing standards for query (including SQL-92, OQL-93, and OQL-93 Basic). The Query Service provides an architecture for a nested and federated service that can coordinate multiple, nested query evaluators.

9.1.1.1.10 Licensing Service

The Licensing Service provides a mechanism for producers to control the use of their intellectual property. Producers can implement the Licensing Service according to their own needs, and the needs of their customers, because the Licensing Service does not impose its own business policies or practices. A license in the Licensing Service has three types of attributes that allow producers to apply controls flexibly: *time*, *value mapping*, and *consumer*. Time allows licenses to have start/duration and expiration dates. Value mapping allows producers to implement a licensing scheme according to units, allocation (through concurrent use licensing), or consumption (for example, metering or allowance of grace periods through "overflow licenses"). Consumer attributes allow a license to be reserved or assigned for specific entities. For example, a license could be assigned to a particular machine. The Licensing Service allows producers to combine and derive from license attributes. The Licensing Service consists of a *LicenseServiceManager* interface and a *ProducerSpecificLicenseService* interface. These interfaces do not impose business policies upon implementors.

9.1.1.1.11 Property Service

Provides the ability to dynamically associate named values with objects outside the static IDL-type system. Defines operations to create and manipulate sets of name-value pairs or

name-value-mode tuples. The names are simple OMG IDL strings. The values are OMG IDL *any*s. The use of type *any* is significant in that it allows a property service implementation to deal with any value that can be represented in the OMG IDL-type system. The modes are similar to those defined in the *InterfaceRepositoryAttributeDef* interface and are designed to be a basic building block, yet robust enough to be applicable for a broad set of applications. The property service provides "batch" operations to deal with sets of properties as a whole. The use of "batch" operations is significant in that the systems and network management (SNMP, CMIP, ...) communities have proven such a need when dealing with "attribute" manipulation in a distributed environment. There are provisions for exceptions such that *PropertySet* implementors may exercise control of (or apply constraints to) the names and types of properties associated with an object, similar in nature to the control one would have with CORBA attributes. *PropertySet* implementors restrict modification, addition and/or deletion of properties (read-only, fixed) similar in nature to the restrictions one would have with CORBA attributes. Finally, the property service provides client access and control of constraints and property modes.

9.1.1.1.12 Time Service

The Time Service enables the user to obtain current time together with an error estimate associated with it, allowing clients to ascertain the order in which "events" occurred. Time-based events are generated with timers and alarms with support for computing the interval between two events. The interface consists of two services: *TimeService* and *TimerEventService*. *TimeService* manages Universal Time Objects (UTOs) and Time Interval Objects (TIOs). The *TimerEventService* manages Timer Event Handler objects.

9.1.1.1.13 Security Service

The security functionality defined by this specification comprises six areas: authentication of principals (human users and objects which need to operate under their own rights) to verify their identity, authorization and access control to decide whether a principal can access an object, security auditing to make users accountable for their security related actions, security of communication between objects requiring trust to be established between the client and target through authentication, integrity protection, and confidentiality protection, non-repudiation to provide irrefutable evidence of actions such as proof of origin of data to the recipient, and administration of security information (for example, security policy).

9.1.1.2 In-Progress Object Services

All of these services are either in the Request For Proposal (RFP) or Request For Information (RFI) stages.

9.1.1.2.1 COM/CORBA Interworking, Part B (RFP)

When adopted, this service will extend the existing COM/CORBA Interworking mappings to include the new release of the Microsoft Distributed COM (DCOM) specification to ensure continued interoperability between the two object models.

9.1.1.2.2 ORB Portability (RFP)

ORB implementation experience demonstrated that some of the ORB Object Adapter specifications were functionally incomplete if required to allow 100 percent code portability across ORBs. The ORB Portability specification generated from this effort will provide a more robust implementation by addressing some of the following issues: object server program activation, assignment of object implementations to servers, creation of implementation definition objects, server and ORB activation rules for object implementations, notification between object adapter and object server when objects are ready to receive requests, synchronization between object implementation and object adapter, object deactivation, activation of object groups, memory management within object implementations, multithreading and critical sections within object implementations, object server shutdown processes, and ORB exception handling.

9.1.1.2.3 Multiple Interfaces and Composition (RFP)

The composition service provides the means for objects to be composed of logically distinct services by the use of multiple interface definitions. To accomplish the goals of the composition service, the requirements for this service include the following list of requirements: multiple service interfaces, version control, session management, fully qualified operation names, flexible subtyping, additional COM model support, and increased object access/visibility.

9.1.1.2.4 Messaging (RFP)

This service provides mechanisms for managing asynchronous messages in distributed object systems. This is accomplished through interface changes which allow (1) clients to make requests on an object without blocking the client execution thread, (2) clients to make requests that may not complete during the lifetime of the client execution environment, (3) clients to establish a mechanism that will receive the response and process it appropriately, (4) clients and servers to specify the quality of service that is to be employed when making or responding to a request, and (5) object servers to control the order in which incoming requests are processed.

9.1.1.2.5 Object Access By Value (RFP)

In CORBA, objects are passed by reference. Operating upon a reference to a remote object is inappropriate for some situations, such as when an object encapsulates a data structure. In these situations, passing the object by value provides for greater efficiency. The Object Access By Value service will add this dimension by allowing objects passed in parameters to operate within contexts. If an object is being passed by value as an "in" parameter, then the caller is the sending context and the recipient is the receiving context. If the object is an "out" parameter, then the caller is the receiving context and the recipient is the sending context. If the parameter is an "inout" parameter, then the roles are meaningful only with respect to a single parameter directional flow. Finally, if the parameter is a return parameter, the caller is the receiving context and the original recipient of the operation is the sending context.

9.1.1.2.6 Java Language Mappings (RFP)

Similar to other language mappings, a Java language mapping will provide transformations from OMG IDL to Java. The proposed effort maps the entire IDL language and provides access to all of the features of the CORBA (especially the DII/DSI and server side mappings). It also includes the mapping of PIDL constructs of the CORBA 2.0 specification so that consistency exists with the IDL mapping.

9.1.1.2.7 Secure Socket Layer (RFP)

The CORBA Security specification defines security in a CORBA system including interoperability, whether or not the underlying transport is secure. The Common Secure Interoperability specification defines a particular security mechanisms for interoperability. It does not address the issue of using a secure transport layer, in particular SSL. So, this effort will result in specifications directing the use of SSL in a secure CORBA environment.

9.1.1.2.8 Analysis and Design (RFP)

The Object Analysis and Design (OA&D) service defines the interfaces and semantics needed to support the creation and manipulation of a core set of OA&D models, that, in turn, define the structure, meaning, and behavior of object applications within the OMG Object Management Architecture. These models may be static models, dynamic models, usage models, architectural models, or other models of value during analysis and design. These services will play a key role in achieving semantic interoperability between OA&D tools, such that users of these tools may be ensured that models created by one tool are meaningful to another tool, that there is no semantic loss when interchanging models, and that import and export issues will be handled correctly.

9.1.1.2.9 Internet (RFI)

The Internet Services Request for Information (RFI) will collect information to help guide the adoption of specifications that will scale the OMG Object Management Architecture to the Internet and further populate or align it with Internet standards, protocols, tools, and utilities. The result may be the development of an Internet Services Architecture that extends the OMG OMA analogous to the way in which the Object Services Architecture and the Common Facilities Architecture affect the architecture.

9.1.1.2.10 Real-time (RFI)

This Real-time Request For Information (RFI) will collect and document information regarding Real-time object-oriented technologies in order to create a Real-time Architecture for the OMG/OMA. Although the primary emphasis is Real-time support for object-oriented applications, this effort will also consider technologies of more general usefulness, that support embedded systems and fault tolerant systems (as long as the technologies are suitable for Real-time use). The resulting Real-timer Services specification should adopt vendor-neutral common interfaces to (1) improve the quality of Real-time systems and to reduce costs by utilizing the CORBA technologies throughout the Real-time community, (2) standardize interfaces for services throughout the OMA, and (3) communicate the requirements of Real-time Systems to the Platform Technical Committee.

9.1.2 OMA Common Facilities (CORBAfacilities)

Common Facilities provide a set of generic application functions that can be configured to the specific requirements of a particular configuration. These are facilities that sit closer to the end-user than object services or ORB interfaces. The OMA intends to use standardization as a method to gain uniformity in generic operations for end-users to configure their working environment. All of the common facilities are in either the Request For Information (RFI) or Request For Proposal (RFP) stage.

9.1.2.1 Financial (RFI)

Financial facilities seek to produce standard interfaces that are specific to financial and business oriented applications. Areas of interest include electronic fund transfer and security services with target domains of insurance, wholesale and retail banking, securities trading, and real estate.

9.1.2.2 Repositories (RFI)

Repository facilities will provide a common framework for defining content models for information, provide standard user and tool interfaces which access the repository, and support any semantic and administrative services required to maintain open access to the repository.

9.1.2.3 Internationalization (RFP)

These facilities enable developers to use the OMA to create information systems or applications in native language, numeric, currency, and cultural formats.

9.1.2.4 Data Interchange (RFP)

The Data Interchange facility supports interoperability between objects. The key elements of the service include mechanisms for exchanging digital data in a uniform way, using data in both internalized and externalized formats.

9.1.2.5 Mobile Agent (RFP)

The Mobile Agent facility supports the need to create massively distributed information systems over Wide Area Networks (WAN). Agents are programs that have an ability to move and have an ability to start executing autonomously. Agents execute in agent execution engines. The ability to move is used to bring the computation unit closest to where it will be utilized instead of some pre-determined server. The agent facility is used to support the mobility of agents and invocation of agent execution engines.

9.1.2.6 Meta Object (RFP)

A meta-object is defined as an object that represents aspects or features of object types. A meta-object is used to define and specify other objects of interest in the information management system. The meta-objects specify these representation concepts so that they may be explicitly created and managed along with the actual objects they represent. A collection of meta-object types will provide the primitive building blocks needed to represent the complex semantics of object models such as business objects or analysis and design models for applications development. The term object schema is defined as a composition of related meta-objects that together represent semantics of a set of related object types. The Meta-Object Facility provides the framework for creating and managing a variety of meta-objects.

9.1.2.7 Printing (RFP)

The print facility handles the management aspect (scheduling, spooling, locating) of print servers and the routing of print jobs with a range of printing requirements from simple documents to production (high volume) printing.

9.1.2.8 Input Method (RFP)

The input method manager facility allows and standardizes the management of input methods for symbolic character sets (e.g. multi-octet Asian characters) on CORBA platforms.

9.2 Glossary of Abbreviations

AFC2N - Air Force Command And Control Network
AMHS - Automated Message Handling System
API - Application Programming Interface
ATO - Air Tasking Order

C2I - Command,Control & Intelligence
C4I - Command, Control, Communications, Computers &Intelligence
CDE - Common Desktop Environment
CF - Canadian Forces
COE - Common Operating Environment
COM - Component Object Model
CORBA - Common Object Request Broker Architecture
COTS - Commercial Off The Shelf
CMS - Configuration Management System
CPU - Central Processor Unit
CUC - Common User Core

DARPA - Defense Advanced Research Projects Agency
DBMS - Database Management System
DCE - Distributed Computing Environment
DCOM - Microsoft Distributed COM
DFS - DCE File Service
DIA - Defense Intelligence Agency
DII - Defense Information Infrastructure
DISA - Defense Information Systems Agency
DMA - Defense Mapping Agency
DSI - Dynamic Skeleton Interface
DTP - Distributed Time Protocol

ESIOP - Environment Specific Inter-Orb Protocol

GCCS - Global Command And Control System
GDA - Global Directory Agent
GIOP - General Inter-Orb Protocol
GIS - Global Information System
GOTS - Government Off The Shelf

IDL - Interface Definition Language
IEEE - Institute Of Electrical And Electronic Engineers
IIOP - Internet Inter-Orb Protocol
IS - Information System

IT - Information Technology

JPL - Jet Propulsion Laboratory
JPO - Joint Program Office

LAN - Local Area Network
LDAP - Lightweight Directory Access Protocol
LES - Leading Edge Services

NAS - Digital's Network Application Support
NTP - Network Time Protocol

OLE - Object Linking and Embeding
OMA - Object Management Architecture
OMG - Object Management Group
ONC -
OSF - Open Software Foundation
ORB - Object Request Broker

POS - Persistent Object Service

RFP - Request For Proposal
RFI - Request For Information
RPC - Remote Procedure Call

SAA - IBM's System Application Architecture
SIPRNET - DISA Secret IP Router Network
SPR - Software Problem Reporting
SSPI - Security Service Provider Interface
STP-10 - TTCP S-Group Technical Panel 10

TBMCS - Theater Battle Management Core Systems
TIO - Time Interval Object
TTCP - The Technical Cooperation Panel

UTO - Universal Time Object
UUID - Universal Unique Identifier

VTC - Video Teleconferencing

WAN - Wide Area Network
WWW - World Wide Web

9.3 COE Security

The COE Security Appendix is Authored by TTCP C3I TP-11 and issued under the Authority of TTCP C3I Group.

With respect to security for Defense COE's, TTCP C3I Group TP10 covers the UK C2I COE Domain Based Approach, but does not mention the US approach as described in the *IATF Information Assurance Technical Framework* (NSA, August 1999).

As stated in Chapter 7 of this document, NSA envisions the development of an Application Security Service Application Programming Interface (API) that can be added to the US DoD Common Operating Environment (COE). Initially, this API will simplify and improve the development of security-enabled Government Off-The-Shelf (GOTS) capabilities. For each security-enabled application NSA envisions DoD planning will promote standards and establish common usage profiles for secure interoperability. These profiles are intended to enable US DoD planners to form interoperability agreements. NSA also envisions US DoD examining infrastructure needs to ensure that applications can use existing infrastructures. TP11 assumes that the APIs will be transparent with respect to the degree of security provided. For example, DoD Logistics systems can use commercial grade security, while C2 and INTEL systems will require Defense-grade security.

TP-11 will track this development and provide input for coalition implications of any US DoD developed APIs. TP11 cautions TP10 that the future of any particular technology that can be used to support a COE at this point in time is, at best, uncertain. This entails that any security developed for such a COE must be transparent with respect to implementation particulars. TP-11 also cautions that commercially available support for this technology will be sufficient for neither Defense C2 nor INTEL systems. History has consistently shown that the market place does not support the development of Defense-grade security. TP11 suggests using an approach to security that:

- Supports architectures that map functionality requirements onto COTS and security requirements onto special purpose high assurance devices. Examples of such devices/technologies/architectures include DSTO Starlight, US NRL Pump, DARPA Security wrappers, NAI Labs Domain Based Security, and the UK Domain Based Approach.

- Supports Information Sharing in an environment where Coalitions may be rapidly forming and dissolving, where Coalition partner's may have different trust relations, and where there may be no overriding hierarchical trust structure.

- Supports D-IW concerns such as sensor deployment and operation requirements, while also being compatible with Intrusion Detection Tools and IW C4ISR tools, such as EPIC, Shapes Vector, and Ironman.

- Accommodates new technologies, such as ATM and mobile code.

- Works over Tactical Networks, which are inherently low bandwidth, error prone, and volatile.

- Is susceptible to assurance arguments and other forms of verification, such as red teaming.

Specific research areas necessary to implement this approach are discussed in TP11's *Secure Information Systems Strategic Assessment* (1996) and *Safe Use of the Internet* (1997) Reports. TP11 will supply TP10 these documents upon request. TP11 POC for this request will be TP11's UKNL.

10. Distribution List

Distribution List for TTCP C3I TP-10 report
Trends and Applications in COEs and DCEs

Published April 2000

Title	Country	Number of documents
C3I Group Counsellor		1
Executive Chairman C3I Group		1
National Leader C3I Group	AS	1
Washington Group Officer	AS	1
Defence Document Repository	AS	1
National Leader C3I Group	CA	1
Washington Contact Officer	CA	1
Defence Document Repository	CA	1
National Leader C3I Group	NZ	1
Washington Contact Officer	NZ	1
Defence Document Repository	NZ	1
National Leader C3I Group	UK	1
Washington Contact Officer	UK	1
Defence Document Repository	UK	1
National Leader C3I Group	US	1
Defense Document Repository	US	1
Chairman C3I-TP-6		1
Chairman C3I-TP-8		1
Chairman C3I-TP-9		1
Chairman C3I-TP-11		1
Chairman and Members C3I-TP-10		
Name	Title	
John Laws	Chair	0
Dr Iain Macleod	AS NL	10
Dr John Robinson	CA NL	5
John Tindle	UK NL	10
Carl DeFranco	US NL	10
Other		
Secretary NATO AC/322 SC/4		7
Secretary NATO AC/322 SC/5		7
Hd IS Division NC3A The Hague		5
Chair CCEB AIS ISME		7
Spare		10